Parallele Simulation großer pulscodierter neuronaler Netze

Zur Erlangung des akademischen Grades

DOKTOR - INGENIEUR

vom Fachbereich Elektrotechnik und Informationstechnik der

Universität Paderborn

- Gesamthochschule -

genehmigte Dissertation

von

Dipl.-Ing. Carsten Wolff

aus Brakel-Gehrden

Referent: Prof. Dr. rer. nat. Georg Hartmann

Korreferent: Prof. Dr.-Ing. Ulrich Rückert

Tag der mündlichen Prüfung:

Paderborn 2000

D14 -

Inhaltsverzeichnis

Kapitel 1 - Einleitung

Seit der Antike neigt der abendländische Mensch dazu, den Sitz seiner Persönlichkeit in einem einzigen Organ zu suchen - dem Gehirn. Rom ist *caput mundi* [Kud99] und der Mensch ist *homo sapiens*. Das *cogito, ergo sum* [Des37] der frühen Aufklärung macht deutlich, wie sehr die Assoziation Mensch-Geist-Gehirn unser Leben bestimmt. Aus dieser hirnzentrierten Betrachtung resultiert ein wesentlicher Teil der Faszination, die von der Hirnforschung ausgeht [Wan00]. Neben dem reinen Erkenntnisgewinn über die Funktionsweise unseres Gehirns gibt es allerdings auch ganz handfeste Forschungsinteressen. Mediziner und Biologen wollen Krankheiten heilen und Defekte beheben. Techniker suchen Inspiration zur Auffindung immer neuer Möglichkeiten, menschliche Tätigkeiten an Maschinen zu delegieren.

Einen Interessenschwerpunkt bilden die perzeptiven Fähigkeiten des Gehirns. Da die Erforschung des Gehirns überwiegend von seiner Sensorik ausgehend erfolgt, liegen über die Funktion des Seh- und Hörsystems umfangreiche Untersuchungen vor, die eine Modellierung aussichtsreich erscheinen lassen [HW62]. Diese Nachbildung bedient sich der künstlichen neuronalen Netze, die sich wiederum entsprechend ihrer Nähe zum biologischen Vorbild klassifizieren lassen [MB98]. Gegenstand dieser Arbeit sind die *pulscodierten neuronalen Netze* (PCNN) - insbesondere in ihrer Anwendung in der computergestützten Bildverarbeitung [JP99], die wiederum den Schwerpunkt der Arbeitsgruppe von Prof. Hartmann bildet, bei dem die vorliegende Abhandlung entstand.

Neuronale Netze bestehen aus Neuronen, die den Nervenzellen des Gehirns entsprechen, und einem Verbindungsnetzwerk. Über die Verbindungen empfangen die Neuronen Eingangsreize von anderen Neuronen und erzeugen daraus einen Aktivitätswert [Zel94]. PCNN-Neuronen setzen diesen Aktivitätswert beim Überschreiten einer Schwelle in ein Aktionspotential um - einen Puls oder Spike [MB98]. Die Neuronen des Sehsystems reagieren jeweils nur auf spezielle Reize in einem speziellen Bereich des Gesichtsfeldes mit einem Spike [HW62]. Untersuchungen in Gehirnen haben jedoch gezeigt, daß mehrere Neuronen, die auf einen gleichartigen über das Gesichtsfeld verteilten, jedoch zusammenhängenden Reiz - z.B. eine durchgehende Linie - reagieren, ihre Spikeaussendung synchronisieren, also mit gleicher Frequenz und Phase feuern [EKG90][ERA89][FvD87][GS89]. Neuronen, die auf einen anderen Reiz reagieren, feuern mit dazu unterschiedlicher Phase. Durch das synchrone Feuern einer Neuronengruppe wird in Gehirnen offenbar verteilte Information ge-

bunden, durch eine unterschiedliche Phase werden Informationen voneinander separiert [Eck94]. Der Zeitpunkt der Spikeemission eines Neurons stellt damit neben der aktuellen Aktivität der Zelle eine weitere Dimension der Informationsrepräsentation dar. Diese Synchronisationsmechanismen sind für künstliche Sehsysteme von großem Interesse [Har91].

In dem an diese Einleitung anschließenden Kapitel 2 werden neben Ausführungen zur Neurophysiologie des Gehirns und zum Aufbau des Sehsystems des Primaten insbesondere die Begriffe im Zusammenhang mit den Synchronisationsmechanismen eingeführt. Darauf aufbauend werden künstliche Modellneuronen vorgestellt, wobei das *Eckhorn Neuron* [ERA89] [ERA90][EDA92], auf dem die Untersuchungen in dieser Arbeit basieren, ausführlich behandelt wird. Außerdem werden einige Beispiele für künstliche neuronale Bildverarbeitungsnetze auf Basis des Eckhorn Neurons dargestellt, die zum einen Gegenstand der im Rahmen der Arbeit durchgeführten Simulationen sind und zum anderen die Besonderheiten solcher Netze deutlich machen. Die spezielle Ausnutzung dieser Besonderheiten unterscheidet die vorliegende Arbeit von anderen Untersuchungen zur parallelen Simulation neuronaler Netze.

Neuronale Bildverarbeitungsnetze bestehen aus einer weitaus größeren Anzahl von Neuronen als andere künstliche neuronale Netze [SSW00]. Da die Neuronen auf bestimmte Reize in bestimmten Bildregionen spezialisiert sind, ist eine sehr große Anzahl von Neuronen zur Behandlung komplexer Szenen notwendig. Im Kontext dieser Arbeit sind mit *großen* Netzen solche mit mehreren hunderttausend Neuronen bezeichnet. Aufgrund dieser Problemgröße scheiden verschiedene Lösungsansätze von vornherein aus. Die für PCNN naheliegenden und im Bereich der akustischen Mustererkennung oft genutzten analogen Hardwareneuronen [HMB92][Ehl99] kommen nicht in Betracht, da maximal einige hundert Neuronen mit vernünftigem Aufwand realisiert werden können. Damit ergibt sich die Notwendigkeit einer diskretisierten *digitalen* Simulation. Die Grundlagen und Verfahren einer solchen Simulation werden im Kapitel 3 dargestellt und kritisch gegenübergestellt. Es zeigt sich, daß klassische Simulationsverfahren für künstliche neuronale Netze nicht übernommen werden können und auch mit speziellen Simulationsverfahren auf kommerziellen Standardrechnern keine Simulationsleistung erreicht wird, die für die Untersuchung realer Szenarien unter realen Zeitbedingungen erforderlich wäre. Daher erscheint die Entwicklung dedizierter Spezialprozessoren sinnvoll. Ein solcher Spezialrechner ist der SPIKE128k [FHJ99][Fra97] [HFS97], der den eigentlichen Ausgangspunkt der vorliegenden Arbeit bildet. In diesem Rechner ist ein optimierter Simulationsalgorithmus konsequent in eine Hardware-Pipeline umgesetzt und prototypisch aufgebaut worden. Der Rechner, der in der Arbeitsgruppe von Prof. Hartmann entstand, erreicht auch als Prototyp, der nicht die Möglichkeiten der aktuellen digitalen Schaltungstechnik ausreizt, eine Simulationsleistung, die weit über den Möglichkeiten kommerzieller Standardrechner liegt. Allerdings erforderte schon der Prototypenaufbau einen erheblichen Implementierungsaufwand. Neben der einer dedizierten Hardware

innewohnenden Inflexibilität lag darin der Grund für die Suche nach einer Möglichkeit, auf Basis von Standardkomponenten eine den Spezialprozessoren vergleichbare oder höhere Simulationsleistung zu erzielen. Da einzelne Standardprozessoren - wie im weiteren gezeigt wird - diese Leistung bei weitem nicht bieten können, lag ein paralleler Ansatz nahe, so daß sich die Aufgabenstellung der vorliegenden Arbeit ergab. Ziel der Arbeit ist die Untersuchung der Grundlagen und Möglichkeiten einer parallelen Simulation von großen pulscodierten neuronalen Bildverarbeitungsnetzen. Dazu werden basierend auf den in Kapitel 2 und 3 beschriebenen Grundlagen und ausgehend von den Erfahrungen mit dem SPIKE128k-System Lösungsvorschläge gemacht und Realisierungsmöglichkeiten aufgezeigt.

Ausgehend von den speziellen Charakteristika der beschriebenen Bildverarbeitungs-PCNN erfolgt im Kapitel 4 die Erarbeitung von grundlegenden Verfahren zur parallelen Simulation, die sich mit den Problemklassen der Lastverteilung sowie der Kommunikation und Synchronisation befassen. Dabei spielt wiederum die reine Größe des vorliegenden Problems eine wesentliche Rolle, so daß Lösungen im Bereich der Datenorganisation und Datenspeicherung gefunden werden müssen. Zudem werden Methoden vorgestellt, die eine kompakte Problembeschreibung mit Hilfe der Neuronale-Netze-Beschreibungssprache MNET (Marburger Neuronale-Netze-Beschreibungssprache) [Möl95][Möl96] mittels eines speziellen Compilers in simulierbare Daten verwandeln und auf ein paralleles System verteilen.

Basierend auf diesen Grundlagen wird in Kapitel 5 die Architektur eines parallelen Simulators vorgestellt, die mit Hilfe des PVM (Parallele Virtuelle Maschine) Systems [GBD94] in Software umgesetzt wurde. Dieser Simulator eignet sich sowohl zum Einsatz auf sogenannten *Workstation-Clustern* oder *Clustercomputern* als auch für eine große Gruppe von Parallelrechnern aus der Klasse der massiv parallelen Systeme. Clustercomputer sind im allgemeinen Standardrechner, die mit Hilfe eines lokalen Netzwerks (LAN) gekoppelt sind. Solche Workstation-Cluster sind insbesondere in Forschungseinrichtungen weitgehend verfügbar. Die Methoden zur Verteilung des neuronalen Netzes auf solche Systeme sind in Form eines MNET-Compilers implementiert worden. Dazu wurde die MNET-Sprache in eine kontextfreie Grammatik überführt und mit Hilfe eines Compiler-Compilers daraus ein Präprozessor erzeugt. Dieser bildet zusammen mit verschiedenen Partitionierungsstrategien das Compilersystem. Mittels dieser Softwarewerkzeuge sind die in Kapitel 2 beschriebenen Beispielnetze simuliert und untersucht worden. Es zeigte sich dabei, daß Softwarewerkzeuge aufgrund ihrer Flexibilität der Analyse der Beispielnetze sehr entgegenkommen, die Simulationsleistung auf konventioneller Hardware jedoch den Ansprüchen realer Szenarien nicht genügen kann. Die auftretenden Engpässe führen zur Formulierung von Anforderungen an eine parallele, auf kommerziellen Komponenten basierende Hardwarelösung.

Eine solche auf digitalen Signalprozessoren (DSP) aufbauende parallele Hardware wird im sechsten Kapitel dieser Arbeit beschrieben und es werden verschiedene Vorschläge zu einer Realisierung gemacht. Neben den Signalprozessoren basiert das Konzept auf kommerziellen

Speicherkomponenten und programmierbarer Logik, so daß eine hohe Flexibilität und eine einfache Handhabbarkeit erreicht werden.

Für dieses Konzept erfolgt in Kapitel 7 eine Leistungsabschätzung, die auf Untersuchungen mit Evaluierungswerkzeugen, Softwaresimulationen und einer prototypischen Hardwarerealisierung beruht. Das Konzept wird dann mit verschiedenen anderen Realisierungen verglichen und in den Kontext der Simulationssysteme für große PCNN eingeordnet. Dabei werden sowohl dedizierte Spezialhardwaresysteme als auch andere Neurocomputer, DSP-Parallelrechner und Softwaresimulatoren herangezogen. Das Konzept wird vor dem Hintergrund der durchgeführten Simulationen und des Vergleichs mit anderen Systemen insbesondere bezüglich seiner Effizienz analysiert und bewertet.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick. In diesem letzten Abschnitt wird versucht, auf Basis der durchgeführten Untersuchungen Empfehlungen für die Entwicklung digitaler Simulationssysteme für große PCNN abzugeben.

Kapitel 2 - Neuronale Netze

Neuronale Netze sind ein Modell des Gehirns [Köh90][Zel94]. Mit ihrer Hilfe sollen Aspekte der unterstellten vernetzten und massiv parallelen Informationsverarbeitung in Gehirnen und Nervensystemen nachgebildet und verstanden werden. Die Bezeichnung *neuronal* ist an den aus dem Griechischen stammenden Begriff des *Neurons* angelehnt, der in der Biologie für eine Nervenzelle steht.

Das Modell des neuronalen Netzes geht davon aus, daß die Informationsverarbeitung in Gehirnen auf einer großen Zahl einfacher Nervenzellen - den Neuronen - und einem diese Zellen verbindenden Netzwerk beruht. Die Inspiration für die verschiedenen Ausprägungen des Modells erfolgt im wesentlichen aus dem Bereich der Neurophysiologie. Die Theorie der neuronalen Netze fällt weitgehend in den Bereich der Informatik und des Konnektionismus [Köh90]. Das Modell dient nicht nur zum Verständnis und zur Erforschung der Mechanismen in Nervensystemen, sondern außerdem zum Auffinden von Lösungen für technische Probleme. Damit sind neuronale Netze auch für die Ingenieurwissenschaften interessant.

Im Rahmen dieser Arbeit steht ein spezielles neuronales Modell - das pulscodierte neuronale Netz (PCNN) - und seine Anwendung in der Bildverarbeitung im Mittelpunkt. In den folgenden Abschnitten werden dazu die Aspekte der neuronalen Thematik eingeführt, die für die im Rahmen der Arbeit durchgeführten Untersuchungen Voraussetzung sind.

Aus dem Bereich der neurophysiologischen Grundlagen werden insbesondere Begriffe eingeführt, die sich mit den Abläufen in einzelnen Neuronen und ihren Verbindungen befassen. Weiterhin wird der grundlegende Aufbau des visuellen Systems im Gehirn von Primaten dargestellt, da spezielle Charakteristika der hier vorkommenden Netztopologien im Rahmen der Arbeit ausgenutzt werden.

In einem weiteren Abschnitt wird ein besonderes Phänomen in Gehirnen dargestellt, das sich nur mit den speziellen Modellneuronen nachbilden läßt, die den Ausgangspunkt dieser Arbeit bilden. Das Phänomen, das als Synchronisation im visuellen System bezeichnet wird, ist für die technische Bildverarbeitung von hohem Interesse.

Nachdem die neurobiologischen Grundlagen gelegt sind, befaßt sich der anschließende Abschnitt mit der Modellierung neuronaler Netze. Dazu werden einige Modellneuronen darge-

stellt, die im Verlauf der Arbeit eine Rolle spielen und zudem die grundlegenden Gemeinsamkeiten der verschiedenen neuronalen Modelle verdeutlichen. Darauf aufbauend wird dann das Eckhorn Modellneuron eingeführt, das den Gegenstand der weiteren Untersuchungen bildet. Dieses Modellneuron ermöglicht die Nachbildung der gewünschten Synchronisationsmechanismen.

Der letzte Abschnitt des Kapitels befaßt sich mit einigen Bildverarbeitungs-PCNN. Diese Netze sind zum einen die Basis der im Rahmen der Arbeit durchgeführten Simulationen und zum anderen können anhand dieser Beispiele die speziellen Charakteristika von Bildverarbeitungs-PCNN dargestellt werden, die im weiteren zur Erreichung einer effizienten Simulation ausgenutzt werden.

2.1 Neurophysiologische Grundlagen

Im Gehirn ist das Unvorstellbare verwirklicht: Materie und Geist gehen dort ineinander über. Es ist erstaunlich, daß der Mensch dieses Phänomen so gelassen hinnimmt. Eigentlich müßte es ihm den Schlaf rauben, ihn nicht ruhen lassen, bis er es verstanden hat. Etwas Spannenderes als Hirnforschung kann es schließlich nicht geben [Wan00].

An dieser Stelle soll ausschließlich die Materie näher behandelt werden. Dazu wird die Funktionsweise der Neuronen und ihrer Verbindungen dargestellt und darauf aufbauend einer der am weitesten erforschten Bereiche des Gehirns erläutert, das visuelle System.

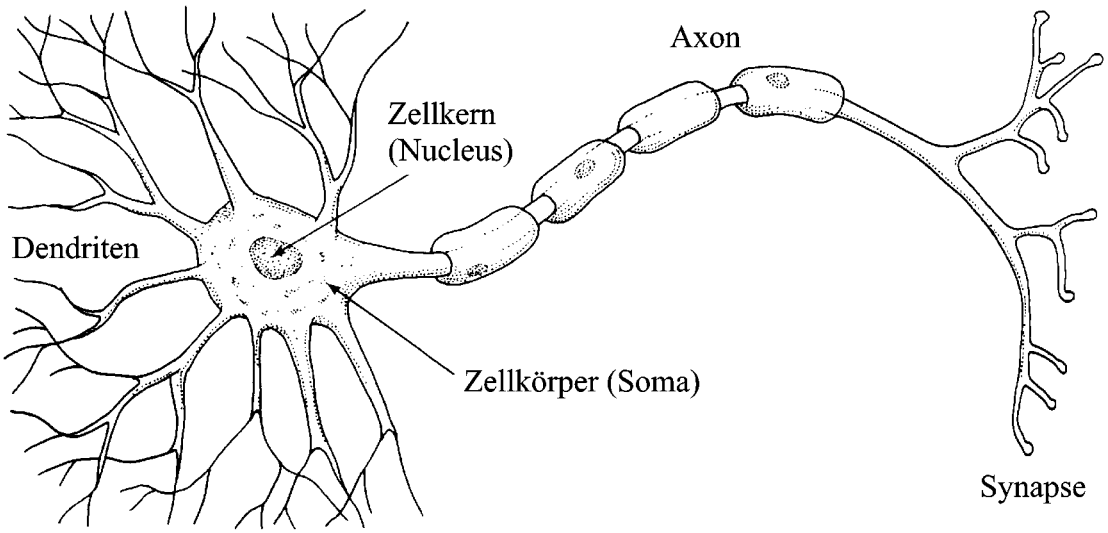

Abb. 2-1 Aufbau eines Neurons (nach [LS91])

Ein Neuron (siehe Abb. 2-1) besteht nach [Hub90] aus *Dendriten*, einem *Zellkörper* und einem *Axon*. Die Dendriten und der Zellkörper empfangen Reizungen durch andere Neuronen,

das Axon überträgt die vom Neuron erzeugten Reizungen an andere Neuronen. Ein Axon kann bis zu einem Meter lang werden, Dendriten werden nur wenige Millimeter lang. Das gesamte Neuron ist mit einer *Zellmembran* umschlossen und mit einer Lösung verschiedener ionisierter Salzmoleküle gefüllt. Das Neuron befindet sich außerdem in einer solchen Salzlösung. Zwischen dem Zellinneren und dem Zelläußeren besteht eine elektrische Potentialdifferenz, auf der die Informationsverarbeitung des Neurons wesentlich beruht.

Im Ruhezustand beträgt die Potentialdifferenz zwischen Intrazellulärraum und Extrazellulärraum bis zu -90 mV. Durch Reizungen der Dendriten und des Zellkörpers durch andere Neuronen verändert sich diese Potentialdifferenz. An den Dendriten bauen sich *dendritische Potentiale (DP)* auf, die sich im Zellkern zum *Membranpotential (MP)* überlagern. Dieser Vorgang erfolgt über spezielle Kanäle in der Zellmembran, die Ionen zwischen Zellinnerem und Zelläußerem austauschen. Durch eine Potentialänderung wird ein weiterer Ionenaustauschprozeß angeregt, der das Ruhepotential wiederherstellt. Wird das Membranpotential jedoch über eine Schwelle hinaus gegenüber dem Ruhepotential ausgelenkt, so setzt ein Prozeß ein, bei dem sehr viele Ionen in das Zellinnere fließen. Damit wird eine positive Potentialdifferenz von bis zu +40 mV zwischen Zellinnerem und Zelläußerem erzeugt. Dieser Zustand wird allerdings nach wenigen Millisekunden durch entgegengesetzte Ionenströme in den Ruhezustand zurückverwandelt. Diese Phase heißt Repolarisation. Dabei wird kurzzeitig sogar eine größere Potentialdifferenz als im Ruhezustand erreicht, d.h. es tritt ein sogenanntes hyperpolarisierendes Nachpotential auf. Der gesamte Vorgang wird als *Aktionspotential* bezeichnet. Ein solches Aktionspotential ist in Abb. 2-2 abgebildet.

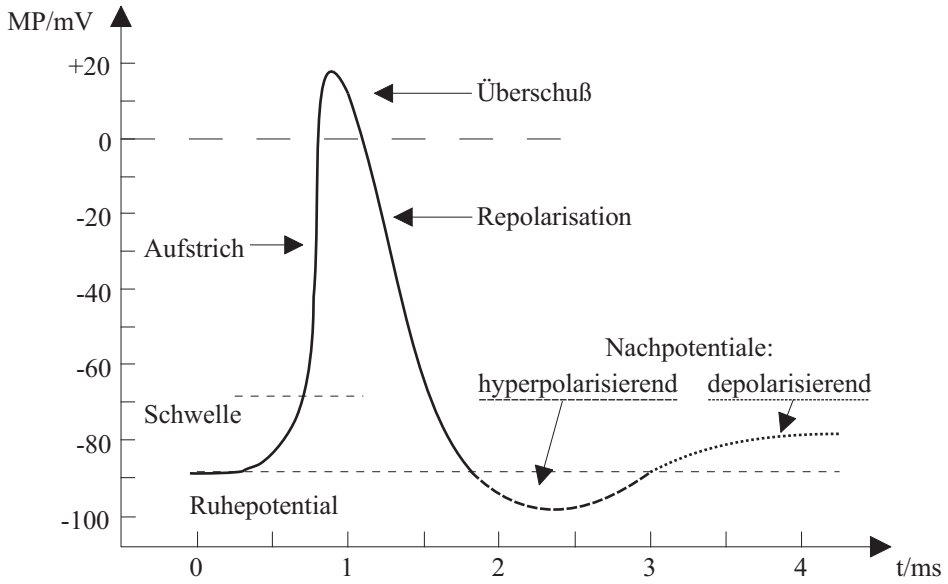

Abb. 2-2 Verlauf eines Aktionspotentials [Köh90]

Dieses Aktionspotential ist das wesentliche Element der Informationsübertragung in Nervensystemen. Es tritt genau dann auf, wenn die Reizung die Schwelle des Neurons über-

schreitet. Die Höhe der Schwelle ist nicht fest, sondern hängt davon ab, wann das Neuron zuletzt ein Aktionspotential ausgesandt hat. Während einer *absoluten Refraktärzeit* kann das Neuron kein neues Aktionspotential erzeugen, anschließend werden die Aktionspotentiale für eine *relative Refraktärzeit* nur in schwächerer Form erzeugt (siehe Abb. 2-3).

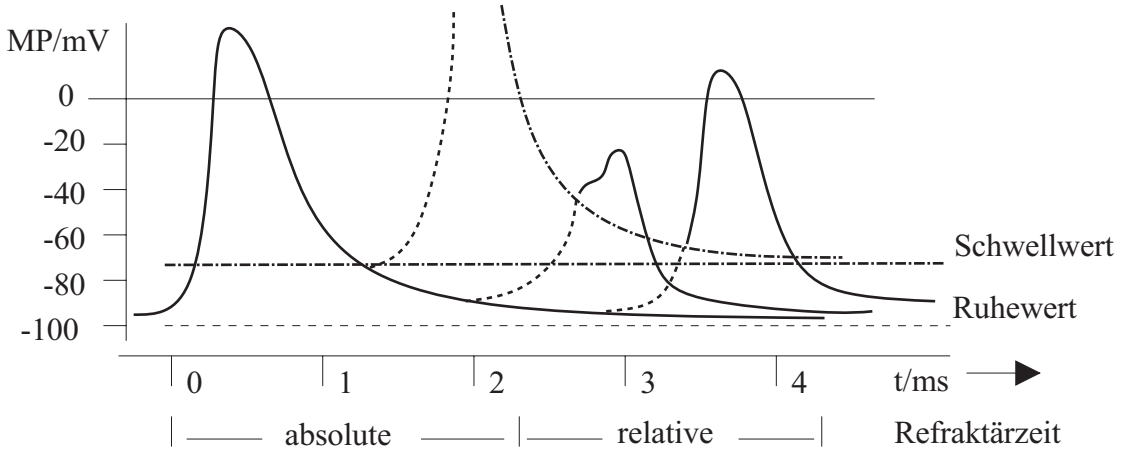

Abb. 2-3 Refraktärperiode eines Neurons [Sch95]

Das Aktionspotential verschwindet nach seiner Erzeugung nicht etwa, sondern es wandert am Axon entlang zu den *Synapsen*. Die Übertragung des Aktionspotentials erfolgt zum Teil chemisch und zum Teil elektrisch. Für eine vollständig elektrische Übertragung ist der Widerstand der Nervenfaser zu groß. Stattdessen wird das Potential wiederum durch das Öffnen und Schließen von Ionenkanälen fortgepflanzt. Da der Ionenaustauschprozeß zur Wiederherstellung des Ruhepotentials an der Zellmembran eine gewisse Zeit benötigt, führen Teile der Zellmembran schon die Repolarisation aus, während weiter entfernte Bereiche sich noch im Aufstrich befinden. Diese Art der Übertragung des Aktionspotentials führt allerdings zu sehr geringen Übertragungsgeschwindigkeiten. Zusätzlich ist daher ein Teil der Axone abschnittsweise mit einer Myelinhülle ummantelt. Diese fettartige Schicht ermöglicht in den entsprechenden Abschnitten eine elektrische Übertragung. Dabei wird das Aktionspotential sehr schnell über die myelinummantelten Bereiche übertragen und an den nicht ummantelten Teilen auf chemischem Wege aufgefrischt. Die Übertragungsgeschwindigkeit eines Aktionspotentials über das Axon hängt damit nicht nur von seiner Länge ab, sondern auch von der Myelinummantelung.

Das Axon verzweigt im Zielgebiet in mehrere Endigungen. Diese Endknöpfchen werden als *Synapsen* bezeichnet (siehe Abb. 2-4). Sie enden direkt an den Dendriten oder am Zellkörper eines anderen Neurons und sind von diesem nur durch einen etwa 20 nm breiten Spalt, den synaptischen Spalt, getrennt. Man bezeichnet das Neuron, dessen Axon in der Synapse endet, als *präsynaptisches* Neuron, das Neuron, das auf der anderen Seite des synaptischen Spaltes liegt, wird als *postsynaptisches* Neuron bezeichnet. Die Synapsen sind mit einer che-

mischen Substanz - einem Neurotransmitter - gefüllt. Sie schütten diese Substanz in den synaptischen Spalt aus, wenn sie von einem Aktionspotential über das Axon erreicht werden. Der Neurotransmitter wird von entsprechenden Rezeptoren am postsynaptischen Neuron aufgenommen und führt dort zum Öffnen von Ionenkanälen, so daß sich ein dendritisches Potential aufbauen kann. Der Neurotransmitter wird dann wieder in den synaptischen Spalt freigegeben und von der Synapse aufgenommen. Durch diesen Vorgang erfolgt die Informationsübertragung unidirektional vom präsynaptischen zum postsynaptischen Neuron.

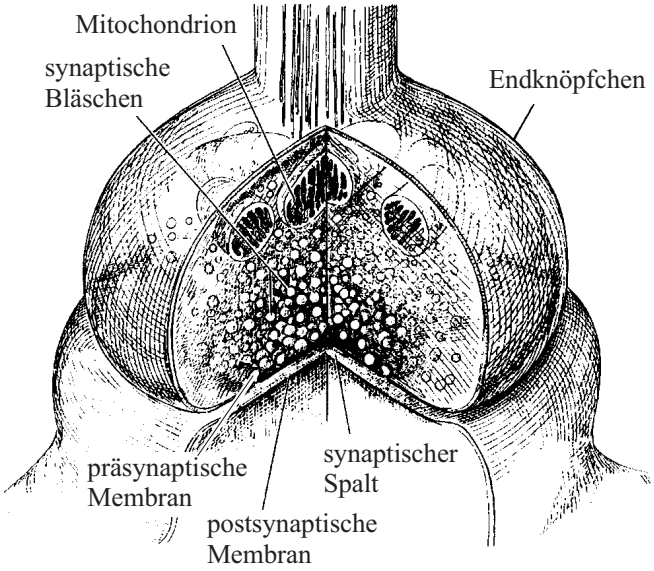

Abb. 2-4 Aufbau einer Synapse (aus [Zel94])

Die Wirkung der Synapsen auf die dendritischen Potentiale kann anregend (exzitatorisch) oder hemmend (inhibitorisch) sein. Exzitatorische Synapsen führen an den Dendriten zum Aufbau einer Potentialdifferenz, ihr Einfluß wird auch *Exzitatorisches Postsynaptisches Potential (EPSP)* genannt. Inhibitorische Synapsen sind in der Lage, ein vorhandenes dendritisches Potential zu verringern, ihr Einfluß wird *Inhibitorisches Postsynaptisches Potential (IPSP)* genannt [Köh90]. Synapsen können zudem auch modulatorisch auf die dendritischen Potentiale wirken, d.h. die Wirkung anderer Synapsen in Form eines solchen Potentials wird multiplikativ verstärkt oder gehemmt [ERA90][EDA92][JP99]. Die durch die Synapsen ausgelösten Effekte sind tatsächlich stark nichtlinear, können jedoch vereinfachend als ähnlich zu einer additiven, subtraktiven oder multiplikativen Änderung der dendritischen Potentiale klassifiziert werden.

Zusammenfassend überlagern Neuronen die Reizungen durch andere Neuronen zu einem Membranpotential, welches beim Überschreiten einer Schwelle zu einem Aktionspotential führt und wiederum über die Synapsen andere Neuronen reizt. Es werden also Aktionspotentiale ausgetauscht, die im weiteren auch *Spike* oder *Puls* genannt werden. Aufgrund dieser Pulse werden PCNN als pulscodiert, pulscodierend oder pulsgekoppelt bezeichnet.

Das visuelle System des Primaten (siehe Abb. 2-6), zu dem in diesem Zusammenhang das Auge in Verbindung mit der Sehbahn zusammengefaßt werden soll, ist gerade auch aufgrund der intensiven Forschungen des Nobelpreisträgers David H. Hubel einer der am besten verstandenen Teile des Gehirns [HW62][Hub90]. Bei der Erforschung dieses Bereichs kommt der Wissenschaft entgegen, daß direkt von einem dem Auge präsentierten Reiz auf die Reaktion eines Hirnareals geschlossen werden kann, was funktionale Zuordnungen erleichtert.

Die Verarbeitung von visueller Information in Form von Lichtreizen beginnt im Auge (siehe Abb. 2-5). Durch das optische System von Iris und Linse fällt das Licht auf einen mehrschichtigen Zellverband, der den hinteren Bereich des Auges auskleidet, die *Netzhaut* oder *Retina*. In der aus Sicht des Lichteinfalls hintersten Schicht besteht dieser Zellverband aus photosensitiven Rezeptoren, den *Stäbchen* und *Zapfen*, wobei die Zapfen Präferenzen für rotes, grünes oder blaues Licht zeigen, während die Stäbchen auf die Lichtintensität insgesamt reagieren.

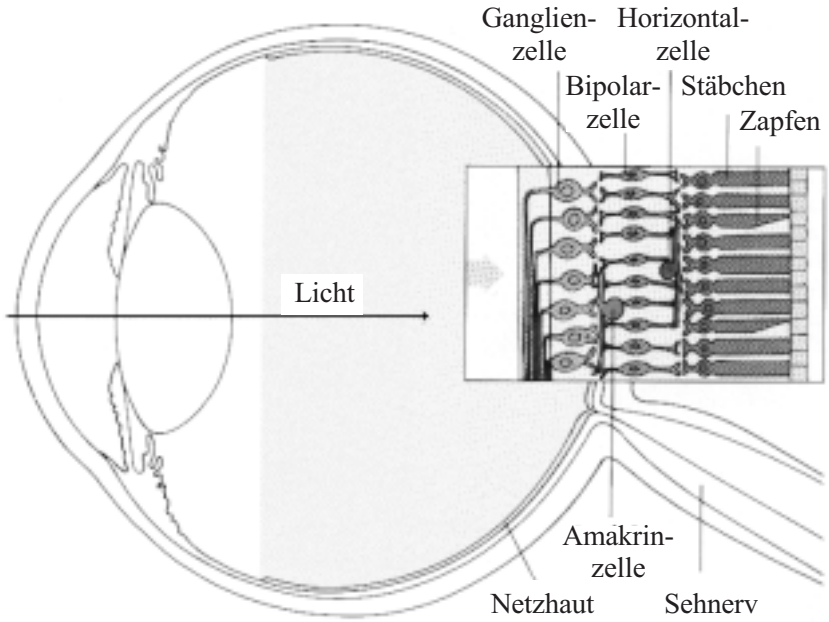

Abb. 2-5 Aufbau des Auges (nach [Hub90])

Die Rezeptoren reizen mit ihrer Aktivität Neuronen aus der Schicht der Bipolarzellen, die wiederum die Ganglienzellen in der nächsten Verarbeitungsschicht anregen. Durch die Horizontal- und Amakrinzellen wird eine weitere Verkoppelung der Horizontal- bzw. Ganglienzellen erreicht. Die Axone der Ganglien bilden den Sehnerv, der durch einen blinden Fleck in der Netzhaut aus dem Auge heraus in das Gehirn geführt wird.

Aus dem schichtweisen Aufbau der Netzhaut läßt sich eine typische Eigenschaft der Neuronen des visuellen Systems erkennen. Die Photorezeptoren reagieren nur auf das Licht, von dem sie getroffen werden, und damit nur auf einen kleinen Ausschnitt des vom Auge aufgenommenen Bildes. Die Neuronen der nächsten Schicht erhalten wiederum nur von Rezepto-

ren in ihrer Nachbarschaft Eingänge, d.h. sie reagieren auf einen etwas aufgeweiteten, aber immer noch begrenzten Bereich, der mit der Position des Neurons in der Schicht korrespondiert. Dieser Effekt bleibt im Laufe der Verarbeitung durch die Neuronenschichten des visuellen Systems erhalten, auch wenn die Bereiche insbesondere aufgrund der lateralen Kopplung z.B. durch die Amakrin- oder Horizontalzellen aufgeweitet werden. Innerhalb einer Schicht von Neuronen sind daher die Zellen entsprechend ihrer Position einem Bereich des durch das Auge empfangenen Bildes zugeordnet, benachbarte Zellen werden durch Reize in benachbarten Bildbereichen angeregt. Die Schichten des visuellen Systems sind *topographisch organisiert*, es handelt sich um eine *topographische Repräsentation* von Information [Hub90]. Der Bereich von Rezeptorzellen respektive Bildausschnitten, der einem Neuron Eingangsreize liefert, heißt *rezeptives Feld (RF)* des Neurons [HW62]. Auf Basis der Zuordnung von partiellen Bildinformationen zu einem Neuron werden diese Informationen durch Koppelung der Neuronen im Gehirn verknüpft und verarbeitet.

Dazu werden die Ausgangspulse der Ganglienzellen über den Sehnerv, den *Nervus Opticus*, zum visuellen Cortex übertragen. Dabei werden unter anderem die Nervenbahnen beider Augen zusammengeführt und entsprechend der Gesichtsfeldhälfte (temporal/nasal), aus der sie Signale verarbeiten, wieder getrennt. Die Verknüpfung der Information beider Augen ermöglicht unter anderem das Tiefensehen mittels der *Stereoopsis*.

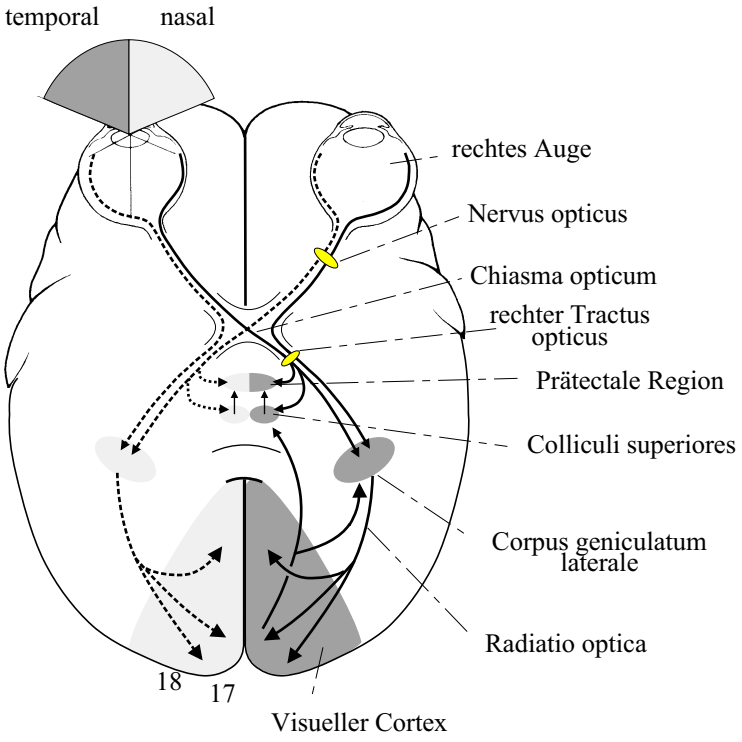

Abb. 2-6 Aufbau der Sehbahn im menschlichen Gehirn (nach [Köh90])

Die Verarbeitung der Bildinformation durch die geeignete Verknüpfung benachbarter Neu-

ronen soll am Beispiel der Ganglienzellen erläutert werden. Ganglienzellen bilden die erste Schicht von Neuronen, die ihre Informationen in Form von Aktionspotentialen, also von Pulsen oder Spikes, ausgeben. Ihre Reizung erfolgt durch die Bipolar- und Amakrinzellen in kontinuierlicher Form. Die Ganglienzellen weisen zwei Formen eines *Zentrum-Umfeld-Verhaltens* auf. Die *On-Zentrum-Ganglien* reagieren nur dann mit gegenüber der Ruhefrequenz erhöhter Pulsfrequenz, wenn ein Lichtreiz auf das Zentrum ihres rezeptiven Feldes fällt und das Umfeld des Zentrums unbeleuchtet bleibt. Wird das gesamte rezeptive Feld gleichmäßig ausgeleuchtet, so unterbleibt eine Reaktion weitgehend, wird nur das Umfeld des Zentrums ausgeleuchtet, so werden die Pulse des Ruhezustandes unterdrückt. Zu erklären ist dieser Effekt damit, daß die Bipolarzellen des gleichen rezeptiven Feldes exzitatorisch mit der Ganglienzelle verbunden sind, während die Zellen benachbarter rezeptiver Felder mit inhibitorischen Synapsen angekoppelt sind. Aufgrund der Überlappung der rezeptiven Felder tritt dann der beobachtete Effekt auf [HW62]. Die zweite Ganglienart, das *Off-Zentrum-Ganglion*, zeigt ein genau entgegengesetztes Verhalten. Lichtreize im Zentrum haben bei ihnen inhibitorische Wirkung, Reizungen in der Peripherie haben exzitatorische Wirkung.

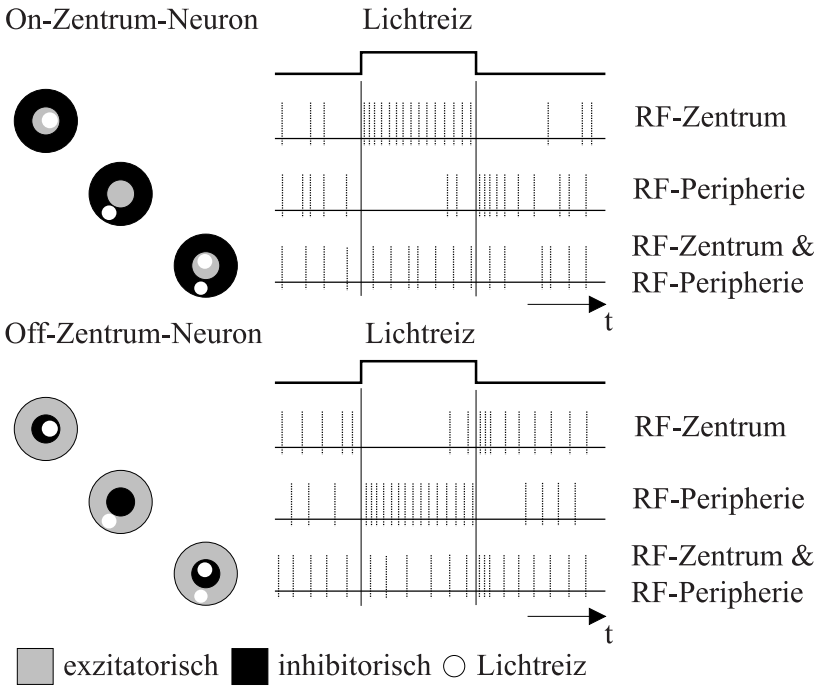

Abb. 2-7 Reaktion der Ganglienzellen auf einen Lichtreiz an unterschiedlichen Positionen in ihrem rezeptiven Feld (nach [Fra97])

In Abb. 2-7 ist links das rezeptive Feld der Neurons mit der Trennung von Zentrum und Peripherie abgebildet und der jeweilige Ort der Reizung durch einen Lichtpunkt gekennzeichnet. Rechts ist jeweils die Art der Spikefolge beim Einschalten und Ausschalten des Lichtreizes aufgetragen. Auf Basis dieses einfachen Zellverhaltens sind die folgenden

Schichten von Neuronen in der Lage, eine immer komplexer werdende Reizspezifität zu entwickeln. Indem die Neuronen nur auf spezielle Reize in speziellen Bildbereichen reagieren, wird durch die Zellschichten des visuellen Systems Information aus den Bildern extrahiert und verteilt repräsentiert. Die Zusammenführung und Verbindung der Informationen mehrerer Neuronen geschieht dabei durch die Kopplungsstruktur der Synapsen. Neben diesem Mechanismus der Trennung und Verbindung von Information ist jedoch noch ein weiteres Phänomen in Hirnstrukturen gefunden worden, mit dem sich der folgende Abschnitt beschäftigt.

2.2 Synchronisation im visuellen System

Durch die verteilte Repräsentation der Information entsteht das Problem der Verbindung von verteilter Subinformation zu einer Gesamtinformation. Im Kontext des visuellen Systems entsteht die Frage, wie aus der Aktivität vieler Neuronen, die sehr spezielle, räumlich begrenzte Eigenschaften einer vom Auge aufgenommenen Szenerie repräsentieren, eine Gesamtwahrnehmung und damit erst ein Verstehen dieser Szenerie entstehen kann. Bis zu Beginn der achtziger Jahre wurde für dieses *Verbinden der Information* im wesentlichen der hierarchische Aufbau des visuellen Systems verantwortlich gemacht. Durch die Verbindungsstruktur sollten alle Merkmale einer komplexen Szenerie im Extremfall auf ein einzelnes Neuron konvergieren, das dann durch seine Aktivität die Wahrnehmung dieser Szene repräsentiert [Bar72]. Diese Theorie wird auch als *Großmutterzellentheorie* bezeichnet, da das Gehirn des Menschen dann über ein Neuron verfügt, das nur reagiert, wenn die Großmutter erkannt wird. Beim Ausfall dieser Zelle würde die eigene Großmutter nicht mehr erkannt. An der Großmutterzellentheorie orientieren sich viele künstliche neuronale Modelle.

Als biologisch plausible Lösung des Problems wurde jedoch ein anderer Mechanismus vorgeschlagen, den die Süddeutsche Zeitung als kleine Revolution in der Gehirnforschung bezeichnet [Rub98]. Als Reaktion auf einen zusammenhängenden Reiz synchronisieren alle Neuronen, die auf spezielle Teilreize dieses Gesamtreizes ansprechen, ihre Spikeausgabe, d.h. sie Feuern mit gleicher Pulsfrequenz und Phase. Die Korrelation von einzelnen Szenenmerkmalen wird durch die Synchronität der diese Merkmale repräsentierenden Neuronen gekennzeichnet [vdM81]. Dieser Mechanismus kann erklären, wie auf Basis lokaler Eigenschaften rezeptiver Felder und einer darauf basierenden verteilten Informationsrepräsentation ein perzeptorischer Gesamteindruck entsteht [Eck94]. Insbesondere im Bereich der neuronalen Bildverarbeitung kann durch die Verbindung der zeitlichen Dimension der Verarbeitung mit der hierarchischen Struktur des Gehirns ein plausibles und leistungsfähiges Abbild der Natur konstruiert werden, mit dessen Hilfe das sogenannte *Binding Problem* gelöst wird [EBJ88][Har91].

Die Reaktion von Zellgruppen, die auch *Assemblies* [Heb49] genannt werden, auf einen Reiz durch synchrones Feuern konnte von verschiedenen Gruppen [FvD87][EBJ88][ERA89] [GS89][EKG90] an Gehirnen von Katzen und Affen nachgewiesen werden. Mit unterschiedlichen Meßmethoden wurde dazu über Elektroden in den Gehirnen der betäubten oder wachen Tiere die Spike-Aktivität von einzelnen Zellen sowie die mittlere Aktivität von größeren und kleineren Assemblies gemessen. Es ließen sich sowohl rhythmische als auch nicht-rhythmische Oszillationen mit Frequenzen von einigen zehn Hertz nachweisen. Insbesondere die nicht-rhythmischen Oszillationen sind offenbar sehr direkt an kurzfristig auftretende visuelle Stimuli gebunden [Eck94].

Die Mechanismen, die zu solchen Oszillationen führen, sind in verschiedenen theoretischen Analysen [DPG94][JE97] und Simulationen untersucht worden [GRH93][RGF93]. Von Interesse war insbesondere die Frage, wie sich die Korrelationen zwischen den Oszillationen von Neuronen oder Neuronen-Assemblies verstärken oder schwächen lassen [EKG90] [JE97]. Neben der Synchronisation von Neuronen zur Verbindung zusammenhängender Merkmale wird die Desynchronisation der Oszillationen zwischen den Assemblies als Mechanismus zur Trennung unterschiedlicher Reize vonaneinander angenommen [KS91] [SK91].

Die Erzeugung der Oszillationen erfolgt offenbar durch sogenannte *Linking-Synapsen,* die Neuronen in einer Zellschicht miteinander verkoppeln. Wird eines dieser Neuronen aktiv und emittiert einen Spike, so werden andere Neuronen, die auf einen ähnlichen Reiz an einer möglicherweise verschobenen Position reagieren, verstärkt angeregt. Wenn diese Neuronen ohnehin aufgrund des ähnlichen Reizes feuern würden, so wird der Zeitpunkt ihres Feuerns aufgrund der Verstärkung über die Linking-Synapsen vorgezogen und somit mit dem Feuern des zuerst überschwellig gewordenen Neurons synchronisiert. Die Kopplung durch laterale Linking-Verbindungen kann dabei auf unterschiedliche Arten erfolgen [Har91][KS91] [JE97], der zu erzielende Effekt ist jedoch ähnlich. Die biologischen Grundlagen dieser Effekte sind weiterhin Gegenstand der Forschung und Diskussion, insbesondere erfolgt die kritische Auseinandersetzung mit einem vielversprechenden Begründungsansatz, der auf einer möglichen modulatorischen Wirkung der Synapsen beruht [ERA90][EDA92][JP99].

Die Erzeugung solcher Oszillationen ist in neuronalen Bildverarbeitungssystemen erstrebenswert. Mit Hilfe der Synchronisationsmechanismen lassen sich Probleme wie die Objekt-Hintergrund-Separation, die Konturverfolgung oder die Repräsentation mehrerer Objekte gleichzeitig durch eine Neuronenschicht [MB98][JP99] lösen. Einige Beispiele für solche Bildverarbeitungsnetze werden im weiteren noch vorgestellt. Der Einsatz eines künstlichen Neuronenmodells, das die Erzeugung der Synchronisationseffekte ermöglicht, und seine Simulation für eine große Anzahl von Neuronen sind daher wichtige Ziele im Bereich der Bildverarbeitung.

2.3 Modellierung neuronaler Netze

Um die geschilderten perzeptiven Fähigkeiten von Gehirnen untersuchen und verstehen zu können und außerdem diese Fähigkeiten in technischen Systemen nutzbar zu machen, ist es erforderlich, die Effekte auf einer anderen Plattform als dem Gehirn zu simulieren. Simulation beinhaltet den Entwurf von Modellen realer dynamischer Systeme zum Zweck der Nachbildung auf Rechenanlagen [Rüc96]. Ein häufig eingesetztes Modell von Gehirnstrukturen ist das neuronale Netz. Die vorliegende Arbeit beschäftigt sich mit Simulationsmethoden und Simulatorarchitekturen für eine spezielle Klasse des neuronalen Modells - mit den *pulscodierten neuronalen Netzen (PCNN)* [MB98]. Da zur Erstellung geeigneter Simulationsmethoden eine vertiefte Kenntnis der Eigenheiten des zu simulierenden Modells notwendig ist, soll in diesem Abschnitt auf die Modellierung neuronaler Netze im allgemeinen sowie im speziellen auf das Eckhorn Modellneuron [ERA89][EDA92], das die Grundlage dieser Arbeit bildet, eingegangen werden.

Neuronale Netze bestehen vereinfachend aus Verarbeitungseinheiten, den Neuronen, und aus einem diese Einheiten verbindenden Netzwerk. Dabei unterscheidet sich jedoch der Grad der Abstraktion und Vereinfachung der unterschiedlichen Modelle stark. Je nach Anwendungszweck variiert die Bandbreite von sehr einfachen formalen Modellen wie den McCulloch&Pitts Neuronen [MP43] bis hin zu biologienahen Modellierungen des einzelnen Neurons wie den Hodgkin&Huxley Modellen [HH52]. Trotzdem soll an dieser Stelle versucht werden, eine Beschreibung einzuführen, die zumindest für die einfachen Modelle bis hin zum biologienahen Eckhorn Neuron nutzbar ist. Mit Hilfe dieser Beschreibung lassen sich die grundlegenden Mechanismen künstlicher neuronaler Netze aufzeigen.

Zur Informationsverarbeitung im künstlichen Neuron werden die von Vorgängerneuronen kommenden Eingangssignale gewichtet und aufsummiert. Das Ausgangssignal des Neurons wird dann aus dieser Summe über die Aktivierungsfunktion erzeugt und eventuell an Nachfolgeneuronen weitergegeben. Dieses Neuron läßt sich entsprechend Abb. 2-8 abstrahieren.

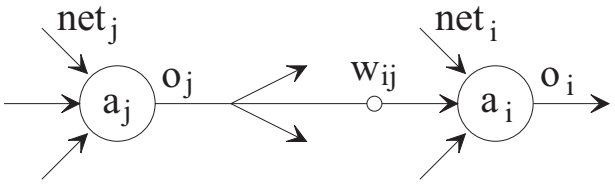

Abb. 2-8 Formalisierte Darstellung zweier Neuronen [Zel94]

Ein künstliches Neuron *i* besteht danach aus den folgenden Komponenten [Zel94].

- Dem *Aktivierungszustand* $a_i(t)$, welcher dem Membranpotential der biologischen Zelle entspricht.

- Der *Aktivierungsfunktion* f_{act}, die angibt, wie sich ein neuer Aktivierungszustand $a_i(t+T)$ des Neurons i aus der alten Aktivierung $a_i(t)$ und der Netzeingabe $net_i(t)$ zusammensetzt. Der neue Zustand errechnet sich somit aus

$$a_i(t+T) = f_{act}(a_i(t), net_i(t), \theta_i(t)), \qquad \text{(Gl. 2.1)}$$

  wobei $\theta_i(t)$ der Schwellwert des Neurons ist.

- Der *Ausgabefunktion* f_{out}, welche die Ausgabe $o_i(t)$ aus dem Aktivierungszustand $a_i(t)$ bestimmt.

$$o_i(t) = f_{out}(a_i(t)) \qquad \text{(Gl. 2.2)}$$

- Einem *Verbindungsnetzwerk*, über welches das Neuron mit anderen Zellen kommuniziert. An jedem Verbindungspunkt des Neurons mit einer anderen Zelle wird das Eingangssignal mit dem Faktor w_{ij} gewichtet. Die Indizes geben die Zellen an, die an dieser Verbindung beteiligt sind. Im Unterschied zur sonst im Bereich der künstlichen neuronalen Netze anzutreffenden Notation handelt es sich bei w_{ij} um die Verbindung der präsynaptischen Zelle j mit der postsynaptischen Zelle i, d.h. der erste Index i bezeichnet das Neuron, zu dem die Verbindung führt. Es lassen sich drei Fälle unterscheiden:

  1. $w_{ij} = 0$: Es besteht keine Verbindung zwischen Neuron j und Neuron i.
  2. $w_{ij} > 0$: Es besteht eine erregende (exzitatorische) Verbindung zwischen Neuron j und Neuron i.
  3. $w_{ij} < 0$: Es besteht eine hemmende (inhibitorische)Verbindung zwischen Neuron j und Neuron i.

- Der *Propagierfunktion*, die angibt, wie sich die Netzeingabe $net_i(t)$ aus den Ausgaben $o_j(t)$ der anderen Neuronen und den Verbindungsgewichten w_{ij} berechnet.

$$net_i(t) = \sum_j o_j(t) \cdot w_{ij} \qquad \text{(Gl. 2.3)}$$

Auf Basis dieser Notation soll im weiteren die Darstellung der neuronalen Modelle erfolgen.

Ein wichtiger Aspekt im Bereich der neuronalen Netze ist die Adaption der Verbindungen des Netzwerkes, die im allgemeinen als *Lernen* bezeichnet wird. Das Lernen ist eine Modellierung der in Gehirnen zu beobachtenden Synapsenplastizität. Im Rahmen der vorliegenden Arbeit wird der Bereich des Lernens nur am Rande betrachtet, da sich die in der Arbeitsgruppe von Prof. Hartmann erstellte Dissertation von Martin Schäfer [Sch00] u.a. mit der Einpassung von Lernverfahren in die vorgestellten Simulationsverfahren befaßt. Die biologischen

Grundlagen des Lernens und die gewählten Modellierungen sind dort ebenfalls ausführlich geschildert. Die daraus resultierenden Methoden und Architekturen [SH99][SSW00] werden an den entsprechenden Stellen berücksichtigt, soweit sie die vorliegende Arbeit betreffen.

Da die in [Sch00] behandelten Lernverfahren auf der verbreiteten Lernregel nach Hebb [Heb49] basieren, soll diese in ihrer Grundidee vorgestellt werden. Die *Hebb'sche Lernregel* läßt sich einem Satz wiedergeben:

> *Wenn das Axon einer Zelle A in der Lage ist, eine Zelle B so anzuregen, daß es wiederholt und anhaltend die Zelle B zum Feuern bringt, dann setzt ein Wachstums- oder Stoffwechselprozeß in einer oder beiden Zellen ein, der die anregende Wirkung von Zelle A auf Zelle B verstärkt.*

Ein solches Verhalten von Synapsen in Gehirnen konnte bisher nicht sicher nachgewiesen werden, die meisten bisher unterstellten Mechanismen der Synapsenplastizität lassen sich aber in Form dieser Regel abstrahieren [MB98].

In künstlichen neuronalen Netzen werden entsprechend dieser Regel die Gewichte der Verbindungen modifiziert. Mathematisch vereinfacht läßt sich die Auswirkung der Regel von Hebb auf die Gewichte w_{ij} durch das Inkrement Δw_{ij} mittels des Zusammenhangs

$$\Delta w_{ij} = k \cdot a_i \cdot o_j \qquad \text{(Gl. 2.4)}$$

darstellen, wobei k oft als Lernrate bezeichnet wird, die zwischen 0 und 1 gewählt wird. Wählt man $k = 0$, so wird nichts gelernt. Es existieren verschiedene Varianten und mathematische Formulierungen dieser Regel [Zel94].

Die Lernregel nach Hebb gehört zu den Verfahren für *unüberwachtes Lernen*, da die Änderung der Gewichte nur von lokalen Parametern der beiden beteiligten Zellen abhängt und kein von außen vorgegebenes Korrektursignal eingesetzt wird, wie es in Form von Lern- oder Fehlermustern bei den *überwachten Lernverfahren* üblich ist. Im Rahmen einer biologienahen Modellierung erscheinen nur unüberwachte Verfahren plausibel, da in Gehirnen keine Korrektursignale von außen auf ein Neuron einwirken [Zel94].

2.3.1 McCulloch&Pitts Neuronen

Entsprechend dem Einsatzzweck variiert die Biologienähe und Komplexität der Modellierungen im Bereich der künstlichen neuronalen Netze. Insbesondere der Konnektionismus arbeitet mit sehr formalen, einfachen Modellneuronen, um die aus der massiven Parallelität und der systematischen Vernetzung der Neuronen resultierenden informationstechnischen Fähigkeiten zu untersuchen und effizient nachbilden zu können. Diese Modelle werden mittels Softwaresystemen auf Digitalrechnern simuliert, so daß ein Zeitschrittverfahren angewandt wird, dem die einfache Modellierung entgegenkommt. Im Jahr 1943 stellten

McCulloch und Pitts ein solches formales Neuron vor [MP43]. Es handelt sich um ein einfaches Schwellwertneuron, das zum Zeitpunkt $t+T$ feuert, wenn zuvor, also zum Zeitpunkt t, der Schwellwert θ von der Summe der gewichteten Eingangssignale $o_j(t)$ überschritten worden ist. Dabei ist T die Zeitschrittlänge. Der Netzeingang des Neurons wird durch

$$net_i(t) = \sum_j o_j(t) \cdot w_{ij} \qquad \text{(Gl. 2.5)}$$

beschrieben. Der Ausgabewert $o_i(t+T)$ errechnet sich über eine einfache Schwellwertbildung direkt aus dem Netto-Input. Es gilt also mit $a_i(t) = f_{act}(net_i(t), \theta) = net_i(t) - \theta$:

$$o_i(t+T) = f_{out}((net_i(t) - \theta))$$

$$\text{mit} \qquad f_{out}(x) = \sigma(x) = \begin{cases} 1 \text{ , falls} & x > 0 \\ 0 \text{ , falls} & x \leq 0 \end{cases} \qquad \text{(Gl. 2.6)}$$

Dadurch, daß der Ausgabewert $o_i(t+T)$ zum Zeitpunkt $t+T$ geliefert wird und die Auswertung der Eingangswerte $o_j(t)$ schon zum Zeitpunkt t erfolgt ist, besitzt dieses Neuron eine Refraktärzeit von $t_{refrak} = T$. Somit kann das McCulloch&Pitts Neuron zu jedem Zeitpunkt $t = kT$, mit $k \in [1 \ldots N]$, innerhalb von N Zeitschritten feuern.

Dieses Neuron ist in seinem Aufbau wesentlich einfacher gehalten als ein natürliches Neuron und auch mathematisch einfacher als die allgemeine Beschreibung. Allerdings eignet es sich durch seinen vereinfachten Aufbau gut, um mathematische Aussagen über die Fähigkeiten neuronaler Netze zu machen. Zudem ist es Grundlage vieler Modelle neuronaler Netze und daher Gegenstand verschiedener Simulationen [Zel94][Res93a][Res93b][Res93c].

Insbesondere lassen sich McCulloch&Pitts Neuronen mit Hilfe einer Matrix-Vektor-Schreibweise darstellen, die für viele einfachere Neuronenmodelle benutzt wird und die üblichen Simulationsverfahren für künstliche neuronale Netze bestimmt [Hee95][KS96]. Dabei werden die Zustände der Neuronen in Form von Vektoren (der Dimension n) notiert, wobei jedes Vektorelement einem Neuron entspricht. Die Verknüpfungsstruktur wird in Form einer Matrix (der Dimension n x n) repräsentiert, wobei die einzelnen Matrixelemente den Verbindungsgewichten w_{ij} entsprechen. Die Simulationsverfahren für solche Netze konzentrieren sich auf die Optimierung der notwendigen Matrix-Vektor-Berechnungen.

Die im Rahmen dieser Arbeiten entwickelten Simulationsmethoden unterscheiden sich davon, da PCNN auf wesentlich komplexeren Modellneuronen beruhen und Effekte simuliert werden sollen, die mit Hilfe der einfachen Modellierung konventioneller neuronaler Netze nicht nachgebildet werden können. Insbesondere im Hinblick auf die Besonderheiten im Bereich der Bildverarbeitung sind daher spezielle Simulationsmethoden notwendig, die sich nur zum Teil an den vorhandenen Methoden im Bereich der neuronalen Netze orientieren.

2.3.2 Spike Response und Integrate&Fire Modelle

Um die Synchronisationseffekte in Gehirnen in künstlichen neuronalen Netzen erzeugen zu können, bedarf es eines Neuronenmodells, das die zeitlichen Abläufe in Nervenzellen insbesondere in Bezug auf die Aussendung von Aktionspotentialen nachbildet. Im Gegensatz zum einfachen McCulloch&Pitts Neuron ist dazu die Modellierung dendritischer Potentiale, die sich über längere Zeit überlagern und abklingen, sowie der Einsatz einer dynamischen Schwelle zur Codierung des Membranpotentials mittels einer Spikefolge notwendig.

Die Klasse der pulscodierten Modellneuronen - auch puslcodierende, pulsgekoppelte oder spikende Neuronen genannt - erlaubt diese Nachbildung der zeitlichen Dynamik. Die dazu notwendige Spikeerzeugung über den Vergleich des Membranpotentials mit einer dynamischen Schwelle wurde von French&Stein vorgeschlagen [FS70]. In diesem Modellneuron wird mittels der exponentiellen Entladecharakteristik von Kondensatoren das Integrations- und Abklingverhalten einer dynamischen Schwelle nachgebildet. Übertrifft die Aktivierung, d.h. das Membranpotential, dieses Neurons den Schwellwert, so wird nicht nur ein Aktionspotential erzeugt, sondern zusätzlich die Schwelle um ein Inkrement erhöht. Dadurch wird das Neuron zunächst an einer weiteren Spikeaussendung gehindert, bis die Schwelle durch exponentielles Abklingen erneut in den Bereich des Membranpotentials absinkt.

Dieses Verhalten ist auch dem *Spike Response* Modell zu eigen, das als Oberbegriff für verschiedene einfache pulscodierte Modellneuronen anzusehen ist [MB98]. Der Aktivierungszustand $a_i(t)$ eines Neurons i, das zu den Zeitpunkten t_i feuert, mit j präsynaptischen Neuronen, die zu den Zeitpunkten t_j feuern, ergibt sich in diesem Modell nach (Gl. 2.7).

$$a_i(t) = \sum_{t_i} \theta(t - t_i) + \sum_j \sum_{t_j} w_{ij} \varepsilon_{ij}(t - t_j) \qquad \text{(Gl. 2.7)}$$

Dabei steht die Summe der $\theta_i(t-t_i)$ für den dynamischen Schwellwert, der direkt die Aktivierung des Neurons verringert. Das exponentielle Abklingverhalten dieses Terms wird durch

$$\theta(t) = -\hat{\theta} e^{-t/\tau} \sigma(t) \qquad \text{(Gl. 2.8)}$$

erreicht, mit $\sigma(t)$ entsprechend (Gl. 2.6). Für $\varepsilon_{ij}(t)$ wird neben anderen denkbaren Verläufen häufig ebenfalls ein exponentieller Verlauf gewählt, wobei über ein zusätzliches Dekrement für t eine axonale Verzögerung der Spike-Übertragung des präsynaptischen Neurons j modelliert werden kann. Ein Spike Response Neuron feuert, wenn der Aktivierungszustand aus (Gl. 2.7) eine feste Schwelle überschreitet. Wird die Summe der $\theta_i(t-t_i)$ als Inkrement in die feste Schwelle einbezogen, so daß $a_i(t)$ nur noch aus der Summe der gewichteten $\varepsilon_{ij}(t)$ besteht, dann wird dieses Modell auch *Dynamic Threshold Model* genannt [MB98].

Ein Beispiel für ein einfaches pulscodiertes Neuron nach diesem Schema ist das *Integrate&Fire Neuron*, dessen Verhalten sich durch eine Parallelschaltung eines Kondensators *C* mit einem Widerstand *R* beschreiben läßt. Aus der Beziehung für den Strom *I*

$$I(t) = \frac{u(t)}{R} + C\frac{d}{dt}u(t) \qquad \text{(Gl. 2.9)}$$

läßt sich durch Einführung einer Zeitkonstanten $\tau_m = RC$ eine Differentialgleichung

$$\tau_m\frac{d}{dt}u(t) = -u(t) + RI(t) \qquad \text{(Gl. 2.10)}$$

aufstellen, die das Verhalten des Spike Response Neurons aufweist, wenn $u(t)$ als Aktivierungszustand des Neurons angenommen wird. Diese Differentialgleichung wird daher bei vielen pulscodierten Modellneuronen als Beschreibung der dendritischen Potentiale verwendet [HMM98][MB98][RD99].

Insbesondere die einfacheren Modelle pulscodierter Neuronen werden zu grundsätzlichen Untersuchungen über die Leistungsfähigkeit dieser Netze und zu Vergleichen mit konventionellen Neuronenmodellen herangezogen [Maa97].

2.3.3 Eckhorn Neuronen

In der Arbeitsgruppe um Prof. Eckhorn an der Universität Marburg wurden im Rahmen der Untersuchungen an Katzen- und Affengehirnen [EBJ88][ERA89] verschiedene Simulationen durchgeführt, für die ein spezielles Modellneuron entwickelt wurde [ERA90][EDA92] [Eck94]. Dieses Modellneuron enthält einige Erweiterungen, die zur Unterstützung bei der Erzeugung der in Kap. 2.2 geschilderten Synchronisationseffekte dienen. Die Modellierung hat sich im Bereich der Bildverarbeitung als vielversprechend erwiesen, so daß das Modellneuron unter der Bezeichnung *Eckhorn Neuron* weite Verbreitung und Eingang in die Literatur gefunden hat [MB98]. Im Rahmen eines gemeinsamen Projektes der Eckhorn-Gruppe mit der Gruppe von Prof. Hartmann ist eine digitale Spezialhardware entstanden, die zur Beschleunigung der Simulation von Netzen dieser Eckhorn Neuronen dient [Fra97] [HFS97][FHJ99]. Dieses Projekt war Ausgangspunkt der vorliegenden Arbeit. Daher beruhen die durchgeführten Untersuchungen auf dem Eckhorn Neuron, sind aber in weiten Teilen auf andere pulscodierte neuronale Netze übertragbar.

Das Eckhorn Modellneuron stellt eine Erweiterung der Spike Response Neuronen dar. Im Gegensatz zu diesen werden die gewichteten Eingangsspikes nicht zu einem Gesamtaktivierungszustand aufsummiert, sondern zu mehreren *dendritischen Potentialen (DP)* oder Teilpotentialen, die dann mit unterschiedlichem Einfluß auf die Gesamtaktivierung wirken.

Einer der Dendritenäste wirkt dabei modulatorisch in Form einer Multiplikation auf einen Teil der anderen Teilpotentiale. Dieser *Linking-Dendrit* dient zur Synchronisation mit lateral verbundenen Nachbarneuronen, indem diese an ihm ein *Linking-Potential (LP)* aufbauen. Die Gesamtaktivierung des Neurons formt sich dann zum *Membranpotential (MP)* $a_{i,MP}(t)$.

$$a_{i,\,MP}(t) \;=\; (a_{i,\,DP1}(t) + \ldots + a_{i,\,DPn}(t))\,a_{i,\,LP}(t) + a_{i,\,DPn+1}(t) + \ldots \qquad \text{(Gl. 2.11)}$$

Die Aktivierung $a_{i,MP}(t)$ wird dem Vergleich mit einer dynamischen Schwelle $\theta'_i(t)$ zugeführt, im Falle einer Überschwelligkeit wird ein Spike ausgelöst.

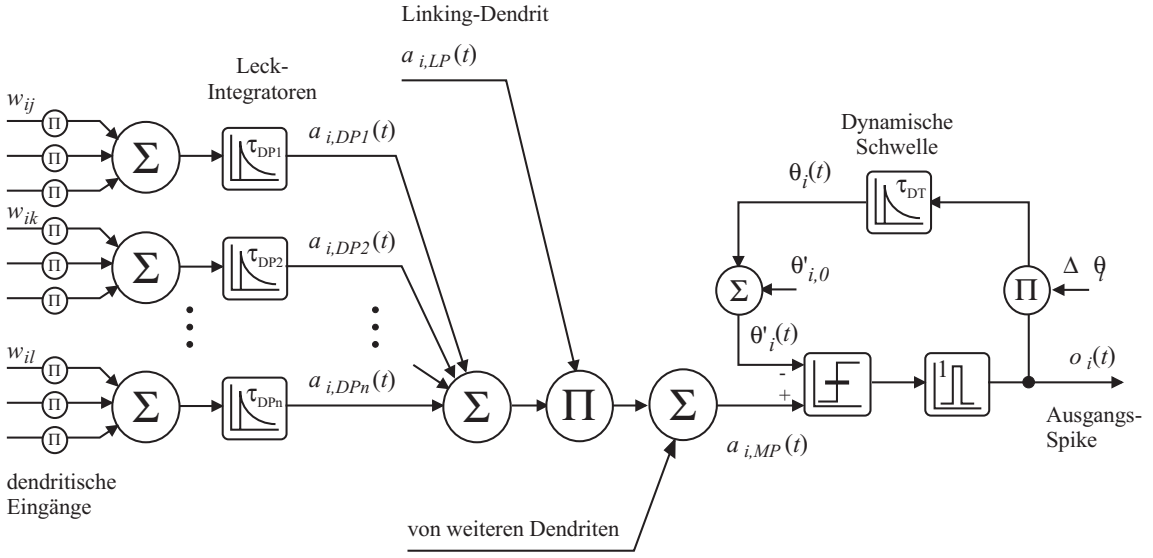

Abb. 2-9 Pulscodiertes Eckhorn Modellneuron (nach [ERA90])

Zum Aufbau eines Modells des visuellen Systems aus Eckhorn Neuronen werden diese Neuronen schichtweise angeordnet. Die Schichten werden miteinander über die additiven dendritischen Potentiale (*DP1* bis *DPn*) entsprechend der Eigenschaften der rezeptiven Felder gekoppelt. Diese Verbindungen werden mit den exzitatorischen Verbindungen des natürlichen Vorbilds gleichgesetzt, d.h. *DP1,...,DPn* entsprechen den exzitatorischen postsynaptischen Potentialen (EPSP) und werden daher im folgenden als *EP1,...,EPn* bezeichnet. Über die Linking-Synapsen erregt ein Neuron das Linking-Potential (LP) benachbarter Neuronen der gleichen Schicht. Die Region, in der die Linking-Synapsen eines Neurons auf andere Neuronen wirken, wird daher auch *Linking-Feld* genannt [Joh94]. Innerhalb des Linking-Feldes kann sich ein Neuron mit seinen Nachbarneuronen synchronisieren. Durch die Überlappung der Linking-Felder erfolgt die Synchronisation des Feuerns ganzer Assemblies. Voraussetzung für eine Synchronisation ist allerdings, daß die benachbarten Neuronen auf einen ähnlichen Reiz reagieren und daher mit einer ähnlichen Frequenz und einer nur leicht verschobenen Phase feuern. Ist dieses für ein Neuron *i* der Fall und ein benachbartes Neuron *j* feuert, so wird über die Linking-Synapse das Linking-Potential $a_{i,LP}(t)$ des Neurons *i* er-

höht. Dieses wiederum führt durch die multiplikative Wirkung zu einer Erhöhung des Membranpotentials $a_{i,MP}(t)$ (siehe Abb. 2-10). Genügt diese Erhöhung zur Überschreitung der dynamischen Schwelle $\theta'_i(t)$, so feuert das Neuron vorzeitig, es kommt also zu einer Spikeverschiebung. Von dieser Verschiebung an feuert das Neuron *i* annähernd synchron zu *j*, wenn beide durch ähnliche Reize ein ähnliches dendritisches Potential $a_{i,DP}(t)$ aufweisen.

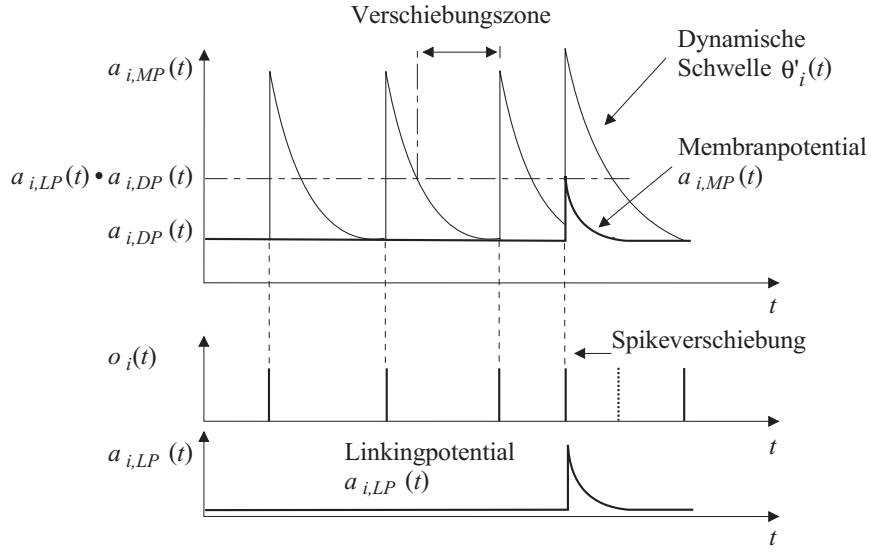

Abb. 2-10 Phasenverschiebung der Spikeaussendung eines Neurons durch eine Erhöhung des Linking-Potentials (nach [Joh94])

Zu beachten ist dabei, daß die Spikeverschiebung nur in einem Zeitfenster (Verschiebungszone) vor der eigentlichen Spikeemission möglich ist, so daß die Reizung der Neuronen über ihre rezeptiven Felder ähnlich sein muß. Durch die Verwendung eines multiplikativen Linking-Potentials wird erreicht, daß Neuronen, die sich im Ruhestand befinden, nicht durch benachbarte Neuronen gereizt werden, wie dies bei einem additiven Linking-Potential der Fall wäre.

Beim Eckhorn Modellneuron erfolgt die Nachbildung der räumlichen und zeitlichen Integration der Eingangsspikes zu den dendritischen Potentialen ähnlich den Spike Response Neuronen über exponentielle Abklingverläufe. Allerdings werden wie schon erwähnt mehrere dendritische Potentiale benutzt, um die unterschiedliche Wirkung der einzelnen synaptisch-dendritischen Verbindungen eines natürlichen Neurons besser nachbilden zu können. Da simulationstechnisch bedingt die Anzahl solcher dendritischen Potentiale begrenzt ist, wird die Netzeingabe $net_{i,DP}(t)$ aus den Eingängen mehrerer Verbindungen gebildet, so daß ein solcher Eingangsbereich auch als Dendritenbaum bezeichnet wird. Das im Rahmen dieser Arbeit verwendete Modellneuron verfügt über vier Dendritenbäume. Zwei dieser Bäume wirken auf das Membranpotential exzitatorisch, d.h. ihre dendritischen Potentiale (*EP1* und *EP2*) werden summiert. Ein weiterer Baum ist der Linking-Dendrit, dessen Linking-Poten-

tial (*LP*) mit der Summe multipliziert wird. Auf das Ergebnis der Multiplikation wirkt der vierte Dendritenbaum subtraktiv, so daß er ein inhibitorisches Potential (*IP*) aufweist. Am Beispiel des Linking-Potentials soll das Verhalten dieser Dendritenbäume aufgezeigt werden, die anderen Bäume verhalten sich entsprechend. Die Netzeingabe $net_{i,LP}(t)$ des Linking-Baumes aus der Erregung durch andere Neuronen ergibt sich zum folgenden Ausdruck.

$$net_{i,LP}(t) = \sum_j w_{ij} \cdot o_j(t) \qquad \text{(Gl. 2.12)}$$

Das dadurch angeregte Linking-Potential $a_{i,LP}(t)$ bildet sich nach der folgenden Differentialgleichung.

$$\tau_{LP} \cdot \frac{da_{i,LP}(t)}{dt} = net_{i,LP}(t) - a_{i,LP}(t) \qquad \text{(Gl. 2.13)}$$

Wendet man auf diese Differentialgleichung die einseitige Laplace-Transformation an, so ergibt sich nach einigen Umstellungen [Ibe99] die folgende Darstellung.

$$A_{i,LP}(s) = \frac{1}{1+\tau_{LP}s} \cdot NET_{i,LP}(s) \qquad \text{(Gl. 2.14)}$$

Somit läßt sich im Zeitbereich die Lösung der Differentialgleichung durch eine Faltung der Netzeingabe mit einer Exponentialfunktion als

$$a_{i,LP}(t) = h_{LP}(t) * net_{i,LP}(t) \qquad \text{(Gl. 2.15)}$$

mit

$$h_{LP}(t) = \frac{1}{\tau_{LP}} \cdot \sigma(t) \cdot e^{-\frac{t}{\tau_{LP}}} \qquad \text{(Gl. 2.16)}$$

darstellen und $\sigma(t)$ entsprechend (Gl. 2.6). Die Lösungen für die anderen Teilpotentiale $a_{i,EP1}(t)$, $a_{i,EP2}(t)$ und $a_{i,IP}(t)$ ergeben sich entsprechend. Das Verhalten der Teilpotentiale wird auch als *Leckintegrator* bezeichnet [MB98]. Auf Basis dieser vier Teilpotentiale ergibt sich das Membranpotential $a_{i,MP}(t)$ zu:

$$a_{i,MP}(t) = (a_{i,EP1}(t) + a_{i,EP2}(t))a_{i,LP}(t) - a_{i,IP}(t) \qquad \text{(Gl. 2.17)}$$

Dabei ist zu beachten, daß entsprechend der Definition des Neurons in [ERA89][ERA90] das Linking-Potential $a_{i,LP}(t)$ über einen Ruhewert gleich 1 verfügt.

Das Membranpotential $a_{i,MP}(t)$ wird mit einer dynamischen Schwelle $\theta'_i(t)$ verglichen, die ähnlich zu den Teilpotentialen über einen Leckintegrator gebildet wird. Grundlage dafür ist wiederum die Beschreibung mittels der Differentialgleichung

$$\tau_\theta \frac{d\theta_i(t)}{dt} = -\theta_i(t) + \Delta\theta_i \cdot o_i(t), \qquad \text{(Gl. 2.18)}$$

wobei sich die Schwelle aus einem veränderlichen Anteil und einem Offset zusammensetzt.

$$\theta_i(t) = \theta'_i(t) - \theta'_{i,0} \qquad \text{(Gl. 2.19)}$$

Bei $o_i(t)$ handelt es sich um die Ausgabefunktion des Neurons, die einen Spike erzeugt, wenn die Aktivierung $a_{i,MP}(t)$ die Schwelle $\theta'_i(t)$ überschreitet. Die Spikeausgabe wird durch eine δ -Funktion ausgedrückt [MB98], so daß sich

$$o_i(t) = \delta(a_{i,MP}(t) - \theta'_i(t)) \qquad \text{(Gl. 2.20)}$$

für die Ausgabefunktion des Neurons ergibt. Die Differentialgleichung für die Schwelle kann ähnlich zu den Teilpotentialen gelöst werden, so daß sich nach der einseitigen Laplace-Transformation mit

$$(1 + s\tau_\theta) \cdot \Theta_i(s) = \Delta\theta_i \cdot O_i(s) \qquad \text{(Gl. 2.21)}$$

über die Umstellung

$$\Theta_i(s) = \frac{1}{\tau_\theta} \cdot \frac{1}{\frac{1}{\tau_\theta} + s} \cdot \Delta\theta_i \cdot O_i(s) \qquad \text{(Gl. 2.22)}$$

nach der Rücktransformation für $\theta_i(t)$ die folgende Faltung ergibt.

$$\theta_i(t) = \frac{1}{\tau_\theta} \cdot e^{-\frac{t}{\tau_\theta}} \cdot \sigma(t) * \Delta\theta_i \cdot o_i(t) \qquad \text{(Gl. 2.23)}$$

Dabei entspricht $\sigma(t)$ wiederum (Gl. 2.6). Durch Einsetzen von (Gl. 2.19) ergibt sich für die Schwelle ebenfalls ein fester Offset $\theta'_{i,0}$. Aus dieser Beschreibung des Eckhorn Neurons lassen sich im weiteren die Simulationsgleichungen für die digitale Simulation herleiten, die Grundlage der in dieser Arbeit behandelten Verfahren sind. Die gewählte Beschreibung mittels Differentialgleichungen erster Ordnung ist heuristisch gewählt [EKG90][Joh94], um das erwünschte Verhalten des Modells möglichst einfach zu erreichen. Andere Modellierungen, insbesondere der Filterfunktionen der dendritischen Potentiale, sind denkbar [Thi99].

2.4 PCNN in der Bildverarbeitung

Der Effekt der Synchronisation und Desynchronisation neuronaler Assemblies verspricht die Lösung verschiedener Probleme in der technischen Bildverarbeitung. In diesem Bereich wird an vielen Stellen versucht, biologisch inspirierte Lösungsansätze in auf Digitalrechnern simulierbare Algorithmen umzusetzen. Insbesondere die weitgehend bekannten Eigenschaften der rezeptiven Felder der frühen Stufen des visuellen Systems werden mit Bildfiltern nachgebildet, um die Extraktion von Bildmerkmalen entsprechend dem biologischen Vorbild zu erreichen. Die zu modellierende Verarbeitung der Sehbahn teilt sich dabei grob in drei Kanäle, die für die Farbbild-, Bewegtbild- und Kontur-Form-Analyse zuständig sind [Hub90]. Im Bereich des Systems zur Kontur-Form-Analyse erfolgt die Dekomposition des Bildes in orientierte Objektkanten oder Linien, d.h. die Neuronen reagieren auf linienhafte Objekte einer bestimmten Orientierung in ihrem rezeptiven Feld.

Dabei stellt sich das Problem, die extrahierten Linienstücke zu einer vollständigen Kontur eines zusammenhängenden Objektes zu verbinden. Als Lösung bieten sich pulscodierte neuronale Netze (PCNN) an, in denen Neuronen, die eine zusammenhängende Kontur repräsentieren, dazu gebracht werden, ihre Spikeausgabe zu synchronisieren. Das synchrone Feuern einer Gruppe von Liniendetektorneuronen zeigt dabei an, daß die repräsentierte Kontur zu einem einzigen Objekt gehört, während das synchrone Feuern einer Neuronengruppe mit anderer Phasenlage ein davon getrenntes anderes Objekt im Bild repräsentiert [Har91] [HD94b]. Die Synchronisation wird dabei über Linking-Felder erreicht. Solche Netzwerke bieten auch bezüglich des Aufwandes Vorteile gegenüber anderen Lösungen [HD94a]. Die Ausbildung der Linking-Felder und der Detektorcharakteristik kann zudem durch Selbstorganisation erfolgen [HD90][CM98].

Einige Beispiele von PCNN, die linienhafte Objekte aus Bildern extrahieren bzw. diese zu Konturen zusammenfügen und vom Hintergrund trennen, werden nachfolgend beschrieben. Diese Beispiele zeigen die speziellen Charakteristika solcher Netze und bilden die Grundlage der im Rahmen dieser Arbeit erstellten Simulationen. Aufgrund der Eigenschaften der Netze werden die vorgestellten Verfahren und Architekturen optimiert, die Simulationsergebnisse werden zur Leistungsabschätzung und Einordnung der Ansätze benutzt. Es handelt sich um zwei Netze aus der Eckhorn-Gruppe und ein weiteres Netz aus der Gruppe von Prof. Hartmann. Anhand dieser Netze werden die grundlegenden Begriffe zur Charakterisierung pulscodierter neuronaler Netze eingeführt. Obwohl ein Leistungsvergleich der verschiedenen Ansätze zur PCNN-Simulation aufgrund der sehr unterschiedlichen Randbedingungen nur begrenzt möglich ist, sollen diese Daten zumindest eine Einordnung der Szenarien und der im Rahmen der Arbeit gewonnenen Ergebnisse in den Kontext dieses noch sehr im Umbruch befindlichen Forschungsgebietes ermöglichen.

2.4.1 Objektseparation nach Weitzel

Im Rahmen eines gemeinsamen Projektes der Gruppe Eckhorn mit der Gruppe Hartmann und diversen anderen Partnern wurde ein großes pulscodiertes neuronales Netz zur Objekttrennung entwickelt [WKS97] und für den im Rahmen des Projektes entstandenen Neurocomputer SPIKE128k [Fra97][HFS97][FHJ99] angepaßt. Dieses Netz dient der Trennung relevanter Objektkonturen vom Hintergrund, der synchronisierten Repräsentation zusammenhängender Konturen und der Desynchronisation mit den Konturen anderer Objekte.

Aus den Eingangsbilddaten des Netzes sind zuvor mit sogenannten *Mexican Hat Filtern*, die den On- und Off-Zentrum Neuronen aus Kap. 2.1 entsprechen, lokale Intensitätskontraste in Form von Kanten extrahiert worden. Die Filterantworten sind hexagonal abgetastet und in Spike-Folgen umgesetzt worden. In der ersten Schicht des Netzes werden dann orientierte Kanten aus dem Bild extrahiert und entsprechend ihrer Orientierung in unterschiedlichen Lagen repräsentiert. Durch Linking-Mechanismen werden unterbrochene Konturen vervollständigt. Aus diesen Kantenrepräsentationen werden durch weitere Schichten Linien-Endpunkte und Objektecken detektiert. Mit Hilfe der Eckendetektoren werden dann die kontinuierlichen, orientierten Objektkanten zu einer vollständigen Objektkontur zusammengefügt. Über Detektoren für T-Übergänge werden unterschiedliche Objekte durch Desynchronisation voneinander getrennt (siehe Abb. 2-11).

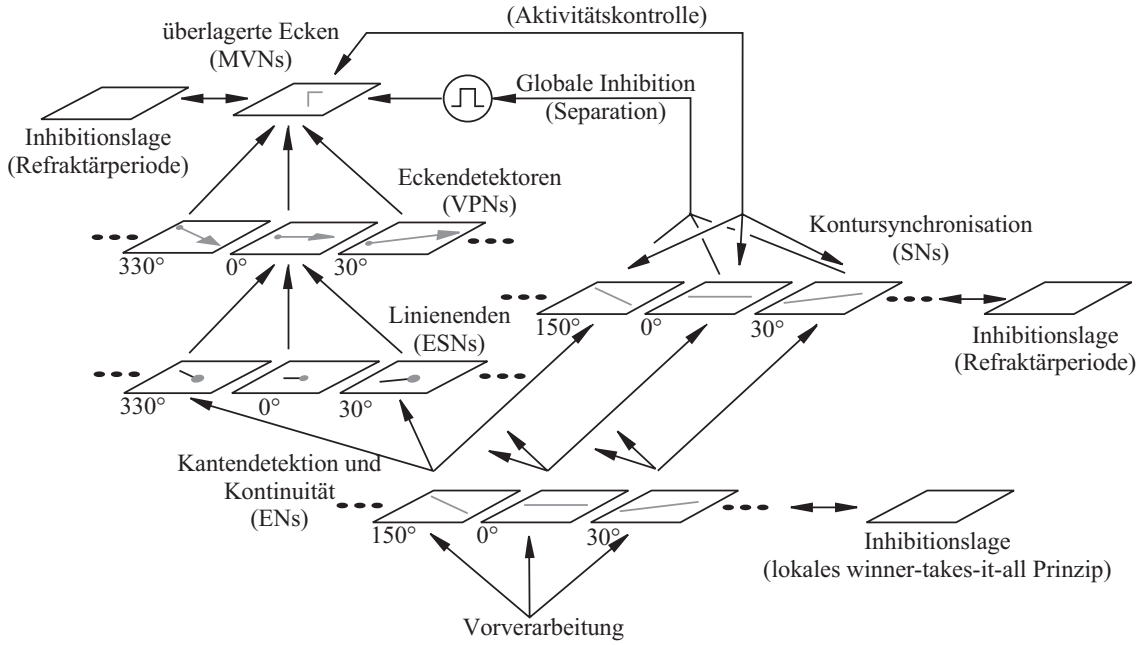

Abb. 2-11 PCNN zur Objektsegmentierung (nach [WKS97])

Das Netz, das im weiteren *Weitzelnetz* genannt wird, ist für eine Simulation auf Digitalrechnern angepaßt, d.h. es wird eine Zeitschrittsimulation vorgenommen. Eine Sekunde der Zeit beim biologischen Vorbild ist in tausend Simulationsschritte unterteilt. Nach einem Ein-

schwingvorgang von etwa 200 Zeitschritten beginnt das Netz, die Konturen des Eingangsbildes zu synchronisieren. Als Eingabedaten wurden zwei Beispielbilder ausgewählt, die jeweils für 2000 Zeitschritte in das Netz eingespeist wurden. Es handelte sich damit um statische Reize, da das Kontur-Form-System des Gehirns ohnehin keine besondere Reaktion für bewegte Reize aufweist.

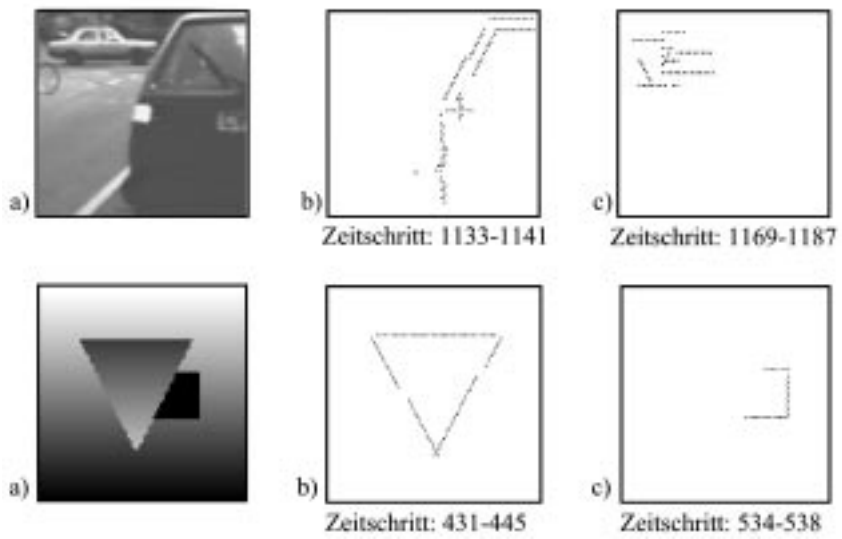

Abb. 2-12 Ergebnisse der Objektseparation durch das Weitzelnetz [WKS97]

Der erste Reiz ist die Aufnahme einer realen Verkehrsszene, in der zwei PKW voneinander getrennt werden sollen. In Abb. 2-12 ist dazu unter a) das Eingangsbild dargestellt. Unter b) und c) sind die feuernden Neuronen der Ausgangslage des Netzes für unterschiedliche Zeiträume abgebildet. Es ist zu sehen, daß während der Zeitschritte 1133-1141 die Neuronen feuern, die den PKW im Vordergrund repräsentieren, während in den Zeitschritten 1169-1187 die Neuronen feuern, die auf den PKW links oben reagieren. Während der Dauer der Simulation feuern diese Neuronengruppen abwechselnd, so daß die PKW voneinander getrennt werden. Beispiel 2 trennt ein teilweise verdecktes Rechteck von einem Dreieck.

Im Rahmen dieser Arbeit sind insbesondere die charakteristischen Daten dieses Netzes von Interesse. Da das Netz auf dem SPIKE128k-System simuliert werden sollte und dieses System maximal 131.071 Neuronen simulieren kann, wurde die Größe der Neuronenlagen entsprechend begrenzt. Das Netz besteht aus 52 Neuronenlagen mit je 2.200 Neuronen sowie einem einzelnen Neuron, so daß das Netz insgesamt 114.401 Neuronen umfaßt. Das Netz wird über zwei Ganglienschichten mit zusammen 4.400 Neuronen gespeist. Die Neuronen sind über insgesamt 3.764.120 synaptische Verbindungen gekoppelt. Die Konnektivität ist weit von einer vollständigen Verknüpfung entfernt (etwa 14 Mrd. Verbindungen), so daß von einer *spärlichen Verknüpfung* gesprochen wird. In Abb. 2-11 ist zu erkennen, daß das Netz systematisch aufgebaut ist. Die Eigenschaften der rezeptiven Felder einer Lage sind gleich, d.h. die Verknüpfungsschemata zwischen zwei Lagen sind für alle Neuronen gleich.

Da jede Lage von Neuronen auf die Detektion einer bestimmten Reizart spezialisiert ist, die meisten Neuronen einer solchen Schicht aber keinen solchen Reiz in ihrem rezeptiven Feld aufweisen, wird nur eine geringe Anzahl von Neuronen erregt. Nur diese Neuronen weisen einen Aktivitätszustand auf, der vom Ruhezustand verschieden ist. Im Weitzelnetz sind im Mittel nur 15% der Neuronen aktiv. Diese geringe Aktivität ist ein wesentlicher Ansatzpunkt für die später erläuterte Simulationsbeschleunigung. Aufgrund der Refraktärzeiten der Neuronen ist nur eine geringe Anzahl der aktiven Neuronen in der Lage, in einem Zeitschritt einen Spike zu erzeugen. Im Beispielnetz feuern im Mittel 0,38% der Neuronen pro Zeitschritt.

2.4.2 Invarianzleistungen nach Stoecker

Ein weiteres zu Simulationen benutztes Beispielnetz stammt ebenfalls aus der Eckhorn-Gruppe. Das *Stoeckernetz* [SER97] (siehe Abb. 2-13) erzeugt aus präsentierten Bildern eine von der Objektgröße und der Objektposition im Bild unabhängige Repräsentation der in einer Szenerie vorhandenen Objekte. Ursprünglich werden diese Repräsentationen einem Assoziativnetzwerk zu Lern- und Erkennungszwecken zugeführt, das im Rahmen der Simulationen in dieser Arbeit jedoch nicht verwendet wurde.

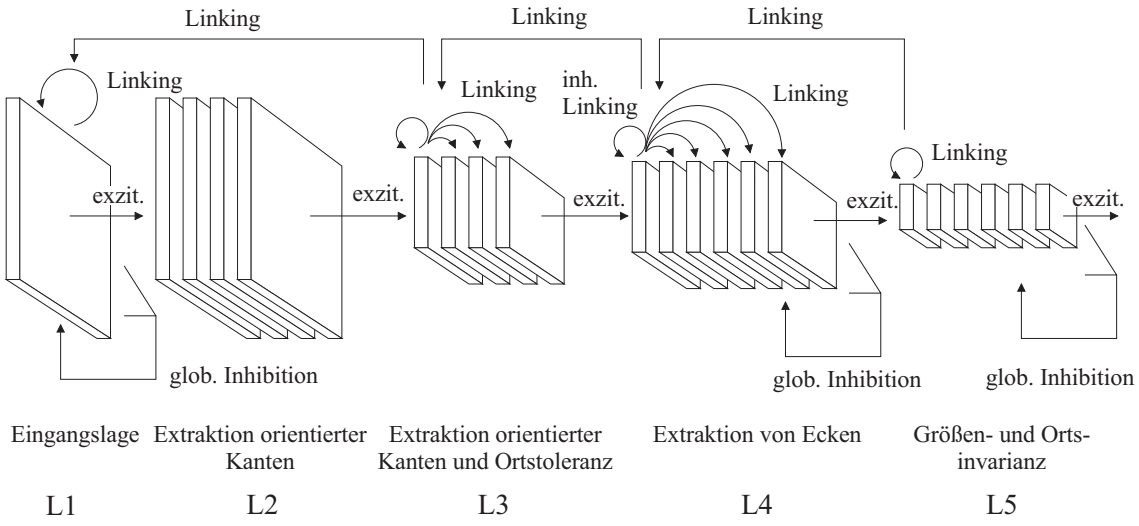

Abb. 2-13 Aufbau eines pulscodierten neuronalen Netzes zur Erzeugung von orts- und größeninvarianten Objektrepräsentationen [SER97]

Das Netz besteht aus fünf Gruppen von Neuronenlagen, von denen allerdings nur die ersten vier Gruppen zur Simulation benutzt wurden. Die erste Lage enthält 64 x 64 Neuronen und extrahiert Konturelemente aus dem Eingangsbild. Diese werden den vier Lagen von Neuronen aus der zweiten Gruppe zugeführt, die auf orientierte Kanten reagieren und ebenfalls 4.096 Neuronen pro Lage enthalten. Die dritte Gruppe von Neuronenlagen reagiert ebenfalls auf orientierte Kanten, jedoch konvergieren mehrere Neuronen aus den korrespondierenden Lagen aus Gruppe 2 auf ein einziges Neuron in Gruppe 3. Durch die so erzielte Aufweitung

der rezeptiven Felder der Neuronen wird eine gewisse Verschiebungstoleranz erzeugt. Die Schichten der Gruppe 3 bestehen daher nur noch aus je 256 Neuronen, die in einem 16 x 16 Raster organisiert sind. Die Schichten der Gruppe 4 reagieren auf die Linienenden der orientierten Kanten aus Gruppe 3. Zusätzlich gibt es in dieser Gruppe zwei Neuronenschichten, die auf die Linienenden zweier orthogonaler Kantenorientierungen reagieren. Jede Neuronenschicht aus Gruppe 4 enthält 256 Neuronen. In den Schichten der fünften Gruppe wird nun die eigentliche Transformation zu einer größen- und ortsinvarianten Repräsentation vorgenommen. Die Neuronen dieser Schichten reagieren auf einen speziellen Winkel der Linie, mit der die Linienenden aus den Schichten in Gruppe 4 verbunden werden können, zu einem Grundwinkel. Diese Winkel sind Merkmale eines bestimmten Objekts, die unabhängig von seiner Position und Größe sind. In den Schichten der Gruppe 5 wird die retinotopische Organisation der Neuronen bezüglich bestimmter rezeptiver Felder aufgegeben. Die Ergebnisse dieser Schicht lassen sich daher sinnvoll nur mit Hilfe des nachgeschalteten Assoziativnetzes analysieren. Zudem haben diese wenigen Neuronen geringen Einfluß auf die Simulationsbedingungen, so daß ihre Simulation wie erwähnt unterblieb.

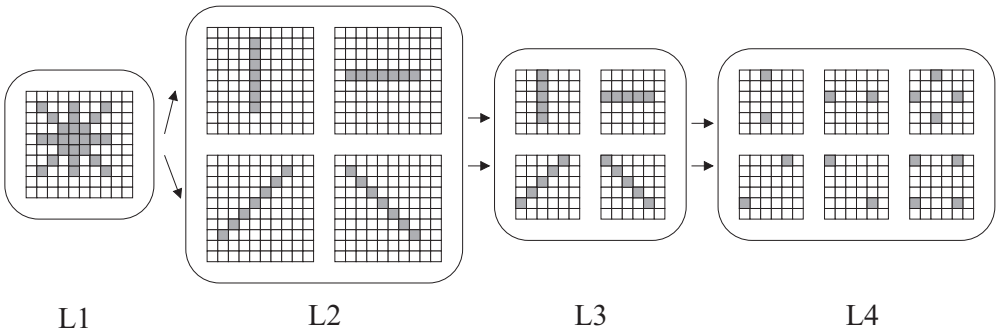

Abb. 2-14 Beispiel der Verarbeitung eines Eingangsbildes durch die Invarianzlagen des Stoeckernetzes (nach [SER97])

Das Stoeckernetz zeichnet sich durch die ausgiebige Verwendung von Linkingverbindungen und den Einsatz einer hohen Zahl unterschiedlicher Verknüpfungsschemata aus. Zudem wird in diesem Netz ähnlich zum Weitzelnetz ein globaler Inhibitionsmechanismus zur Aktivitätssteuerung in den einzelnen Neuronenlagen verwendet. Dazu werden jeweils alle Neuronen einer Lage an jedes einzelne Neuron der Lage über den inhibitorischen Dendritenbaum des Modellneurons angeschlossen. Dieses Verfahren führt zu einer sehr hohen Anzahl von Verbindungen. Das Stoeckernetz verfügt in seiner im Rahmen dieser Arbeit simulierten Form über 16 Lagen (inklusive einer vorgeschalteten Rezeptorlage) von Neuronen mit zusammen 27.136 Neuronen und ist damit deutlich kleiner als das Weitzelnetz. Die Neuronen sind über insgesamt 6.466.182 Verbindungen miteinander verbunden. Die mittlere Aktivität bei Benutzung des in Abb. 2-14 dargestellten Beispielreizes beträgt 50,2%. Es werden im Mittel von 2,1% der gesamten Neuronen Spikes emittiert.

2.4.3 Gaborverhalten mittels PCNN

Die beiden vorgestellten Beispielnetze nach Weitzel und Stoecker sind entworfen worden, um spezielle Eigenschaften der pulscodierten neuronalen Informationsverarbeitung zu demonstrieren. Um eine noch simulierbare Netzgröße zu erhalten, sind dabei einige Vereinfachungen bezüglich der zur Kantenextraktion dienenden Schichten gemacht worden. Diese Vereinfachungen sind im Fall der Beispielszenarien tolerierbar, führen jedoch in komplexeren Szenarien zu Schwierigkeiten. Im Rahmen der technischen Bildverarbeitung werden deshalb zur Merkmalsextraktion Gabor-Filterbänke benutzt, die verschiedene Vorteile bieten [Teu96]. Untersuchungen an cortikalen Zellen haben zudem ergeben, daß die Eigenschaften der rezeptiven Felder dieser Nervenzellen dem Verhalten von Gabor-Filtern entsprechen [Dau80][Mar80]. Daher liegt es nahe, ein Modell der frühen Bildverarbeitung von Retina und Sehbahn zu entwickeln, das zum einen geeignete Verbindungsmuster zwischen den einzelnen Zelltypen aufweist und zum anderen die pulscodierte Verarbeitung des Gehirns nachahmt. In der Gruppe Hartmann ist ein solches Netz entwickelt worden, das im weiteren als *Brausenetz* bezeichnet werden soll [TWH00]. Das Netz ist aus mehreren Schichten von Neuronen aufgebaut (siehe Abb. 2-15).

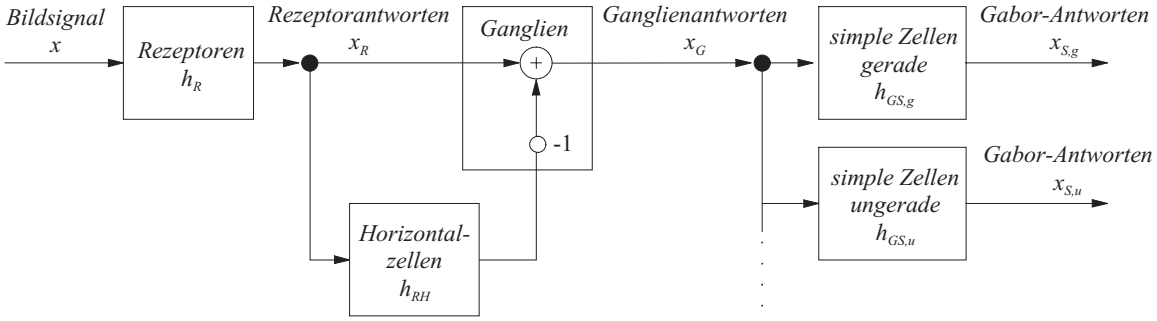

Abb. 2-15 Aufbau eines pulscodierten neuronalen Netzes zur Erzeugung von Gabor-gefilterten Bildrepräsentationen [TWH00]

In einer ersten Rezeptorschicht wird das Bild hexagonal abgetastet und tiefpaßgefiltert. Die Rezeptorantworten dienen dann entsprechend dem biologischen Vorbild als Eingang der Horizontal- und Ganglienzellen. Die Antworten der Ganglienzellen weisen an Grauwertübergängen im Bild Werte ungleich Null auf, sie extrahieren also Grauwertkanten aus dem Bild. Die Ganglienantworten werden dann mehreren Schichten von orientierungsselektiven Neuronen zugeführt, die zugleich die erste Stufe der cortikalen Verarbeitung bilden und *simple Zellen* genannt werden [Hub90]. Die simplen Zellen reagieren mit dem ungeraden oder dem geraden Anteil der Gabor-Filterantworten. Außerdem sind Schichten für sechs verschiedene Linienorientierungen vorhanden, so daß insgesamt zwölf Schichten von simplen Neuronen notwendig sind. Durch geeignete Einstellung der Verknüpfungsschemata zwischen den einzelnen Stufen des Netzes wird erreicht, daß die Antworten der simplen Zellen mit hoher Ge-

nauigkeit den Anworten von Gabor-Filtern entsprechen. Damit ist ein Modell entwickelt worden, das aufzeigt, wie das den Gabor-Filtern ähnliche Verhalten der frühen cortikalen Bildverarbeitung zustande kommen kann [Thi99]. Um das Modell an die pulscodierte Verarbeitung des biologischen Vorbildes anzupassen, sind jedoch einige weitere Schritte notwendig. Im allgemeinen wird angenommen, daß die Ausgänge der Ganglienzellen die erste Stufe der pulscodierten Informationsübertragung bilden, während die vorgelagerten Zellen mit kontinuierlichen Reizungen kommunizieren [Hub90]. Da Pulsfrequenzen im Gegensatz zu Filterantworten keine negativen Werte annehmen können, erfolgt im Modell eine Aufteilung in zwei Pfade (Abb. 2-16), deren Pulsfrequenzen dem negativen und dem positiven Anteil der Gangliensignale entsprechen. Die Anzahl der Schichten der simplen Zellen wird aus diesem Grund ebenfalls verdoppelt, so daß je eine Schicht den positiven Anteil bzw. den negativen Anteil der entsprechenden Gaborantwort repräsentiert. Für die simplen Zellen werden Eckhorn Neuronen mit einem exzitatorischen und einem inhibitorischen Dendritenbaum benutzt. Durch eine synaptische Kopplung der Ausgänge der positiven und der negativen Ganglienzellen mit dem exzitatorischen oder dem inhibitorischen Eingang der Eckhorn Neuronen wird die Verknüpfung der positiven und der negativen Signalanteile erreicht.

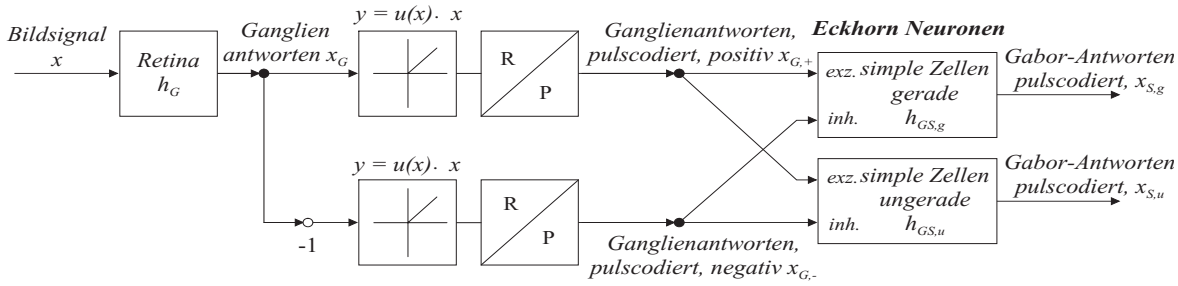

Abb. 2-16 Anpassung des Brausenetzes an die pulscodierte Signalverarbeitung ausgehend von den Ganglienzellen [TWH00]

Im Rahmen der vorliegenden Arbeit ist der pulscodierte Teil des Netzes simuliert worden, so daß im wesentlichen die simplen Zellen nachgebildet wurden. Die Umsetzung der kontinuierlichen Aktivitätszustände der Ganglienzellen in Pulsfolgen erfolgte über die Spike-Encoder und die dynamische Schwelle der Eckhorn Neuronen, eine Modellierung der dendritischen Potentiale war für diesen Zelltypus nicht notwendig. Durch die Aufteilung der simplen Zellen in solche für den positiven Signalanteil und solche für den negativen Anteil verdoppelt sich die Zahl der Zellschichten auf 24. Jede dieser Schichten besteht aus 119 x 119 Eckhorn Neuronen (14.161). Damit umfaßt das gesamte Netz 339.864 Neuronen. Zusätzlich werden 28.322 Spike-Encoder für die Ganglienzellen benötigt. Die Ganglien sind über 20.873.985 Verbindungen mit den simplen Zellen verbunden. Im Durchschnitt werden von 1,6% der Ganglien pro Zeitschritt Spikes emittiert. Diese führen dazu, daß 78,9% der simplen Zellen aktiv sind. Diese hohe Zahl aktiver Zellen ist darauf zurückzuführen, daß in

den frühen Schichten des visuellen Systems die Spezialisierung der Zellen noch nicht sehr ausgeprägt ist und somit sehr viele Neuronen einer Schicht einen ihrer Präferenz entsprechenden Reiz erhalten. Von den simplen Zellen emittieren im Durchschnitt 0,2% pro Zeitschritt einen Spike. Die Aktivitätswerte wurden für einen sehr komplexen Eingangsreiz ermittelt, der in Form eines in der Bildverarbeitung häufig eingesetzten Fotos einer realen Szenerie vorlag [Pla72].

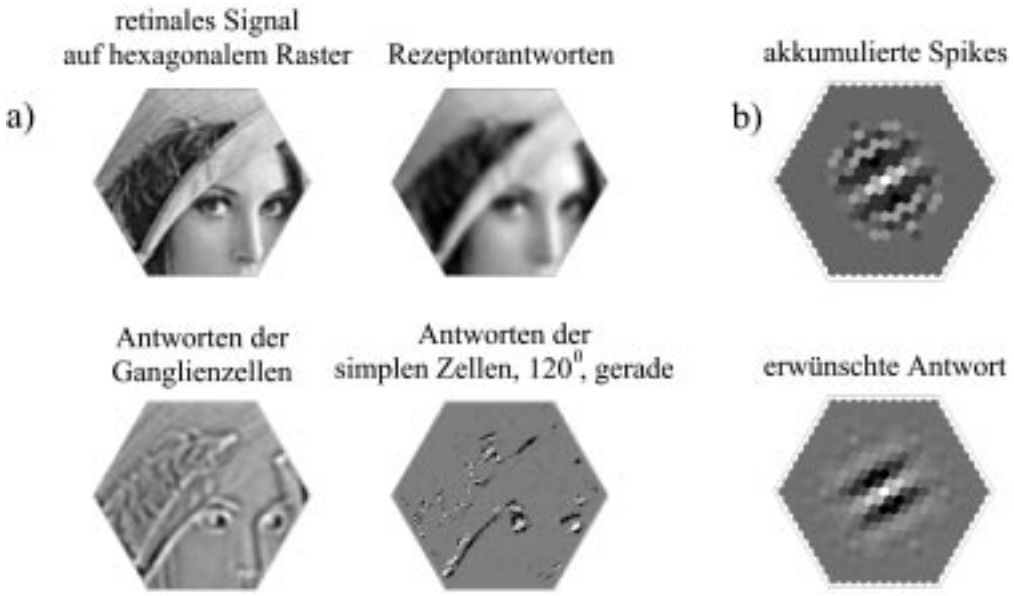

Abb. 2-17 Verarbeitung eines visuellen Reizes a) durch das Brause-Netz und Vergleich der erzielten Impulsantwort b) mit der Impulsantwort im Fall einer kontinuierlichen Verarbeitung [TWH00]

Zusätzlich ist in Abb. 2-17 die durch Akkumulation ermittelte Impulsantwort des Netzes für die auf den geraden, auf 120^o orientierten Anteil spezialisierte Schicht von simplen Zellen angegeben. Diese Antwort ist der Impulsantwort gegenübergestellt, die erzielt wird, wenn keine pulscodierte Verarbeitung gewählt wird. Es ist zu erkennen, daß die gewünschte Charakteristik auch bei einer pulscodierten Verarbeitung entsteht.

2.4.4 Besonderheiten von Bildverarbeitungs-PCNN

Von üblicherweise simulierten und untersuchten PCNN [Maa97][MB98] unterscheiden sich die im Mittelpunkt dieser Arbeit stehenden Bildverarbeitungs-PCNN insbesondere durch ihre Größe von mehreren hunderttausend Neuronen und mehreren Millionen Verbindungen. Die digitale Simulation großer pulscodierter neuronaler Bildverarbeitungsnetze erfordert spezielle Simulationsmethoden, die sich die Besonderheiten dieser Netze zunutze machen. Diese Besonderheiten, die schon exemplarisch anhand der vorgestellten Beispielnetze aufgezeigt wurden, sollen nachfolgend zusammengefaßt werden [SSW00].

Die Kommunikation in PCNN basiert auf dem Austausch von Spikes. Im Gegensatz zu konventionellen künstlichen neuronalen Netzen (KNN), z.B. McCulloch&Pitts Neuronen, ist zur Erzeugung dieser Spikes unter Berücksichtigung des zeitlichen Verhaltens im biologischen Vorbild ein wesentlich aufwendigeres Modellneuron notwendig. Der Anteil an Funktionalität des eigentlichen Neurons bezogen auf die Gesamtfunktion eines neuronalen Netzes ist damit bei pulscodierten neuronalen Netzen wesentlich größer als bei konventionellen Neuronenmodellen.

Übliche Simulationsmethoden für neuronale Netze, die Gruppen von Neuronen durch Vektoren und das Verbindungsnetz in Form von Matrizen darstellen, sind für pulscodierte neuronale Netze wenig geeignet, da der Zustand eines Neurons nicht alleine durch seinen aktuellen Aktivitätswert dargestellt werden kann. Damit scheiden die konventionellen Simulationsmethoden, die auf der beschleunigten Durchführung der Matrixoperationen und dem Austausch von Aktivitätsvektoren beruhen, für die PCNN-Simulation aus und es ergibt sich die Forderung nach einem neuen Simulationsparadigma. Pulscodierte neuronale Netze für die Bildverarbeitung weisen zudem Spezifika auf, die aus der konkreten Anwendung resultieren und bei der Konzeption eines Simulationssystems Berücksichtigung finden müssen.

Bildverarbeitungsnetze basieren auf der Abtastung der Bildinformation durch eine Retina. Die Neuronen des Netzes reagieren jeweils nur auf Reize in einem begrenzten Retinaareal, dem rezeptiven Feld. Daraus resultiert, daß für jeden Neuronentyp im Netz je ein Neuron für jedes rezeptive Feld existieren muß. Ein Bildverarbeitungsnetz ist also aus Lagen gleichartiger Neuronen aufgebaut, wobei jedes Neuron der Lage mit einem Bildbereich korrespondiert, das Netz ist topographisch organisiert. Die Neuronenlagen bilden damit den jeweiligen Verarbeitungszustand des Bildes auf die Aktivitätsverteilung in der Neuronenlage ab. Dieser Aufbau bedingt, daß auch die Verknüpfungsschemata, mit denen die Neuronen mit anderen Neuronen verbunden sind, innerhalb einer Lage ähnlich sind.

Der Aufbau neuronaler Bildverarbeitungsnetze stellt sich als sehr systematisch oder regulär dar. Ein solches Netz ist im Gegensatz zu konventionellen neuronalen Netzen bei weitem nicht vollständig verbunden. Vielmehr sind Neuronen zumeist nur mit den Neuronen des gleichen rezeptiven Feldes in benachbarten Neuronenlagen verbunden. Die Verknüpfung kann damit als spärlich angesehen werden. Viele Neuronen sind mit einem einheitlichen Verknüpfungsschema mit den Zielneuronen verbunden.

Neuronale Bildverarbeitungsnetze reagieren auf relevante Informationen in den auf die Retina projizierten Bildern. Zudem werden diese Informationen separiert und von Neuronenlagen verarbeitet, die auf konkrete Merkmale spezialisiert sind. Da jede Lage Neuronen für jedes rezeptive Feld beinhaltet, wird jeweils nur die geringe Anzahl von Neuronen in einer Lage aktiv, in deren rezeptivem Feld ein solcher spezifischer Reiz auftaucht. Dieses führt zu einer insgesamt sehr geringen Anzahl von aktiven Neuronen in pulscodierten Bildverarbei-

tungsnetzen. Da aktive Neuronen ihre Aktivität in eine Spikefolge umsetzen und der jeweils nächste Spike erst nach einer Refraktärzeit des Neurons erzeugt werden kann, ist die Zahl der Spikes sehr gering. Im Fall einer diskreten Zeitschrittsimulation feuern sehr wenige Neuronen sehr selten, so daß von einer geringen Spikerate gesprochen werden kann. Die Aktivität der Neuronen wird zusätzlich zur Refraktärzeit auch noch durch Inhibitionsneuronen gesteuert. Diese Neuronen hemmen typischerweise eine ganze Gruppe von Neuronen in Abhängigkeit von der Aktivität in dieser Neuronengruppe.

Die Lernvorgänge in Bildverarbeitungs-PCNN erfolgen nach Regeln, die der Lernregel nach Hebb ähneln. Diese Regeln lassen ein unüberwachtes Lernen zu, das nur auf dem Zustand der zu verändernden Synapse und der beiden mit der Synapse verbundenen prä- bzw. postsynaptischen Neuronen basiert.

Zusammenfassend lassen sich PCNN in der Bildverarbeitung durch

- eine sehr hohe Anzahl von Neuronen (10^6-10^7) und Verbindungen (10^7-10^8),
- einen systematischen Aufbau aus topographisch organisierten Neuronenschichten,
- weitgehend reguläre und sich wiederholende Verknüpfungsschemata,
- eine spärliche Verknüpfung (~ 10-100 Verbindungen pro Neuron),
- eine geringe Netzaktivität (~ 20%) und geringe Spikerate (~ 0,5%),
- Inhibitionsneuronen zur Aktivitätssteuerung und
- unüberwachte Lernverfahren, die lokal arbeiten,

charakterisieren [SSW00]. Effiziente Simulationsverfahren und Hardwarearchitekturen entstehen unter Ausnutzung dieser Charakteristika.

Kapitel 3 - Diskrete ereignisgesteuerte Simulation

Pulscodierte neuronale Netze (PCNN) stellen ein interessantes Forschungsgebiet mit vielversprechenden Anwendungen dar. Der Zugang zu diesem Gebiet wird jedoch schon durch die reine Problemgröße und die Komplexität des zu modellierenden biologischen Vorbildes erschwert. Insbesondere der mathematischen Beschreibung und Analyse steht diese Komplexität im Wege. In solchen Fällen entsteht der Bedarf nach Simulationen. Zu unterscheiden sind dabei zwei Motivationen für die Simulation von PCNN. Zum einen soll sie die Analyse und das Verständnis der Modelle sowie auch des biologischen Vorbildes befördern und erleichtern. Außerdem besteht aber auch der Wunsch, die gewonnenen Erkenntnisse in technischen Systemen nutzbar zu machen. Sowohl aus dem Interesse an der Simulation zu Analysezwecken als auch insbesondere zu Zwecken der technischen Nutzung entsteht der Bedarf, die PCNN in realen Szenarien unter Realzeitbedingungen zu simulieren.

Simulation befaßt sich mit der Nachbildung von Vorgängen auf Basis von Modellen der die Vorgänge hervorrufenden Szenarien [Rüc96]. Modelle wiederum sind vereinfachte strukturierte Beschreibung eines Szenarios, die zumeist die als wichtig erachteten Zusammenhänge approximieren und generalisieren. Die Modelle lassen sich in materielle und abstrakte Modelle unterscheiden, wobei sich die abstrakten Modelle im technischen Bereich vorzugsweise mittels einer mathematischen Beschreibung darstellen lassen. Eine mathematische Beschreibung eröffnet zur Untersuchung die Möglichkeit der Anwendung analytischer Verfahren und digitaler Simulationen.

Im Bereich der PCNN sind neben digitalen Simulationen auch Realisierungen der Modelle in Form analoger Hardware [HMB92][MB98][Ehl99] anzutreffen. Diese Beispiele können als materielle Modelle klassifiziert werden. Sie basieren auf der Nachbildung des gewünschten Verhaltens der Neuronen und Synapsen durch analoge Schaltungstechnik und der Kombination dieser Komponenten zu einem neuronalen Netz. Durch die Realisierung jeder einzelnen Komponente in Form einer eigenen Schaltung ermöglichen diese analogen Implementierungen sehr hohe Geschwindigkeiten. Indem sie die massive Parallelität des biologischen Vorbildes aufnehmen, bilden sie die auf den ersten Blick konsequente Umsetzung auf ein technisches System. Tatsächlich wird dieser Ansatz daher in Fällen verfolgt, in denen

eine sehr hohe Geschwindigkeit gefordert ist. Das Problem einer analogen Realisierung liegt in der sehr begrenzten Anzahl simulierbarer Neuronen und Verbindungen, da analoge Schaltungen nicht die hohen Speicherdichten der Digitaltechnik erreichen. Durch die Begrenzung auf aktuell einige hundert Neuronen pro Bauelement [Ehl99] lassen sich mit dieser Technik nicht die für die Bildverarbeitung benötigten Netzgrößen erreichen.

Aus diesem Grund bildet die digitale, zeit- und wertediskrete Simulation großer PCNN den Ausgangspunkt dieser Arbeit. Eine solche Simulation bietet die Möglichkeit, den rasanten Fortschritt der digitalen Schaltungstechnik auszunutzen. Außerdem lassen sich durch die Umsetzung der Verfahren in Form von Software auf kommerziellen Standardrechnern Ergebnisse und Aussagen mit wesentlich weniger Aufwand gewinnen, als dieses mit materiellen Modellen möglich ist. Der eigentliche Hauptgrund für eine digitale Simulation liegt jedoch darin, daß insbesondere durch die hohen Kapazitäten digitaler Speicher die Untersuchung der Millionen von Elementen umfassenden pulscodierten Bildverarbeitungsnetze erst möglich wird [Fra97].

Die digitale Simulation zeichnet sich durch ein diskretes Zeit- und Wertemodell aus. Digitalrechnersysteme basieren auf diskreten Symboltransformationen [Rüc96], so daß die Darstellung der Simulationsdaten in Form solcher diskreten in der Genauigkeit begrenzten Symbole notwendig ist. Zudem erfordert die Methode der Symboltransformationen eine diskrete Zeitfortschreibung, d.h. das simulierte System wird nur zu bestimmten diskreten Transformationszeitpunkten betrachtet. Bezüglich dieser Zeitpunkte kann ein synchrones Zeitmodell, in dem die Zeiträume zwischen den Aktualisierungen des simulierten Systems fest sind, und ein Modell mit dynamischer Zeitfortschreibung, in dem Aktualisierungen nur abhängig von bestimmten Faktoren durchgeführt werden, unterschieden werden. Aufgrund der hohen Werteauflösungen in modernen Digitalrechnersystemen wird der Wertediskretisierung im allgemeinen weniger Aufmerksamkeit gewidmet als der Zeitdiskretisierung, die sich durch die gewählte Feinheit bzw. Zeitschrittlänge direkt auf die erzielten Resultate und auf die Simulationszeit auswirkt [Rüc96]. Dabei konkurriert die Güte der Resultate häufig mit der erzielbaren Simulationsgeschwindigkeit.

Im Zusammenhang mit der Simulation von PCNN hat sich eine besondere Art der digitalen Simulation - die diskrete ereignisgesteuerte Simulation - bewährt [Wat94][Fer95][GA98], die in den folgenden Abschnitten näher beleuchtet werden soll. Dazu wird zunächst die Simulationsmethode im allgemeinen und am konkreten Beispiel der PCNN erläutert und anderen Methoden der digitalen Simulation neuronaler Netze gegenübergestellt. Im Anschluß daran werden die zur Simulation des Eckhorn Neurons benötigten Simulationsgleichungen und der darauf basierende Algorithmus aus den in Kap. 2.3.3 aufgeführten mathematischen Beschreibungen hergeleitet. Darauf aufbauend wird ein sequentielles Verfahren zur PCNN Simulation mit Optimierungs- und Simulationsergebnissen vorgestellt, das den Ausgangspunkt und die Grundlage der vorliegenden Arbeit bildet.

3.1 Methoden zur Simulation künstlicher neuronaler Netze

Zur digitalen Simulation künstlicher neuronaler Netze ist in den letzten zwei Jahrzehnten eine große Zahl von Arbeiten durchgeführt und publiziert worden. Der überwiegende Teil dieser Forschungen befaßt sich mit der Simulation der verschiedenen konnektionistischen Theorien [Hee95][KS96][Zel94] oder aber mit der stark am biologischen Experiment orientierten Untersuchung einzelner Zellmodelle [BB94]. Die in diesen beiden Feldern angewandten Methoden unterscheiden sich recht deutlich voneinander.

Im Bereich der Neurobiologie und der verwandten medizinischen und biologischen Disziplinen werden Simulationen angewandt, um das Verhalten einzelner Nervenzellen oder kleiner Zellverbände bezüglich ihrer elektrochemischen Eigenschaften sehr genau nachzubilden. Diese Nachbildung erfolgt durch die Modellierung selbst einzelner Zellteile (compartments) durch aufwendige partielle Differentialgleichungen, so daß der wesentliche Simulationsaufwand in der iterativen numerischen Lösung dieser Gleichungen liegt. Ein Beispiel für eine recht einfache Modellierung in diesem Bereich ist das Hodgkin&Huxley Modell [HH52]. Da die Simulationen zu Untersuchungszwecken durchgeführt werden, bestehen in diesem Bereich keine Realzeitanforderungen an die Simulation. Mittels dieser Methode werden höchstens einige hundert Neuronen simuliert, wobei z.T. speziell programmierte Simulationen oder aber spezielle Softwaresimulatoren verwendet werden. Ein bekannter Softwaresimulator in diesem Bereich ist *GENESIS* [BB94].

Aus der Sicht der Informationstechnik und Informatik weitaus interessanter sind die Simulationen im Bereich der künstlichen neuronalen Netze (KNN), mit denen meist die konnektionistischen Modelle gemeint sind [Köh90][Zel94]. Diese Netze basieren auf sehr einfachen Modellneuronen wie den in Kap. 2.3.1 beschriebenen McCulloch&Pitts Neuronen und versuchen die durch die vernetzte und massiv parallele Verarbeitungsweise entstehenden informationstechnischen Möglichkeiten zu nutzen. Simuliert wird in diesem Bereich vor allem die Verarbeitung von Mustern durch das Netz, d.h. die Berechnung der Netzausgabe zu einem angelegten Muster, und die Adaption der Verbindungsstruktur auf Basis angelegter Muster, die als Lernen bezeichnet wird. Insbesondere das Lernen beruht auf der Änderung der Verbindungsgewichte in kleinen Schritten, so daß sehr viele Iterationsschritte für einen solchen Lernvorgang nötig sind. Ein bekanntes Verfahren in diesem Bereich ist der Backpropagation-Algorithmus [Zel94][Dit94]. Zur Simulation werden die Aktivitätszustände der Neuronen, die Trainingsmuster und die Netzausgaben häufig in Form von Vektoren dargestellt, die Verknüpfungsstrukturen werden als Matrix notiert [KS96]. Die Simulation der Netze erfolgt dann durch Matrix-Vektor-Berechnungen. Der Aufwand zur Berechnung der Aktivität eines einzigen Neurons oder des Zustandes einer einzigen Verbindung reduziert sich dabei auf wenige Multiplikationen und Additionen. Die Leistungsfähigkeit entsteht erst

durch die Verknüpfung und den Austausch der Daten. Die digitale Simulation solcher künstlichen neuronalen Netze, die üblicherweise einige hundert Neuronen umfassen, erfolgt mit Hilfe von Softwaresystemen auf kommerziellen Standardrechnern [KS96] oder Parallelrechnersystemen [PPD89][NS92]. Digitale Spezialhardware beruht auf der Beschleunigung der Matrix-Vektor-Operationen [Ram92].

Die digitale Simulation pulscodierter neuronaler Netze bewegt sich zwischen diesen beiden Arten der Simulation. Zum einen ist das Eckhorn Neuron nicht so aufwendig modelliert wie die Neuronen der neurobiologischen Simulationen, zum anderen läßt es sich jedoch nicht so einfach in Form einer Matrix-Vektor-Schreibweise darstellen wie die McCulloch&Pitts Neuronen. Während die sehr biologienahen Modellierungen den Hauptaufwand bei der Berechnung des einzelnen Neurons erzeugen, muß bei den abstrakten konnektionistischen Modellen überwiegend die Kommunikation zwischen den Neuronen simuliert werden. Der Aufwand zur Simulation großer PCNN ist gegenüber den konnektionistischen Modellen zu einem wesentlich höheren Berechnungsaufwand für das einzelne Neuron hin verschoben, wenn auch dieser Aufwand nicht den der biologienäheren Modelle erreicht. Tatsächlich handelt es sich bei der PCNN-Simulation um die Berechnung einer großen Anzahl gekoppelter Differentialgleichungen. Zusammen mit der spärlichen Verknüpfung innerhalb der PCNN, die beim Einsatz einer Matrix-Vektor-Repräsentation zu einer sehr dünn besetzten Verknüpfungsmatrix führen würde, und dem geringen Anteil von aktiven Neuronen ergibt sich daher die Anforderung, die Neuronen und Verbindungen einzeln und individuell zu simulieren. Da es zudem aufgrund der sehr hohen Neuronenanzahl nicht möglich ist, jedes Neuron und jede Verbindung auf einer eigenen Recheneinheit zu simulieren, werden PCNN zumindest in Teilen durch sequentielles individuelles Berechnen der Neuronen und Verbindungen simuliert [Fra97][MB98][JRS98]. Ziel der Simulation ist die weitgehend dem zeitlichen Verhalten des biologischen Vorbildes entsprechende Erzeugung der Spikeausgabe der Neuronen und der Reaktion der anderen Neuronen auf diese Spikes. Die Spikes oder Pulse der Neuronen sind damit die relevanten Ereignisse einer solchen Simulation.

Die Methode der diskreten ereignisgesteuerten Simulation verwendet diese Ereignisse zur Reduktion des Simulationsaufwandes, indem Berechnungen nur durch Ereignisse angestoßen werden. Damit wird vermieden, daß Berechnungen durchgeführt werden, die zu keiner Änderung des Ergebnisses der Simulation führen. Exemplarisch entspricht dieses Vorgehen einem Prozeß, der nur auf eine Änderung eines Meßwertes reagiert, während ein zeitgesteuerter Prozeß den Meßwert in regelmäßigen Abständen abfragt. Ein Ereignis (event) symbolisiert damit ein Vorkommnis im realen System, das möglicherweise eine Zustandsänderung nach sich zieht [Rüc96]. Die ereignisgesteuerte Simulation wird auch als *event driven* bezeichnet, die zeitgesteuerte als *time driven* [Fer95]. Die ereignisgesteuerte Simulation basiert historisch auf der Warteschlangentheorie, die in theoretischer Form die Bedienung einer Anzahl von Kunden durch eine Anzahl von Bedienern entsprechend der Ankunfts- und Warte-

zeiten der Kunden beschreibt [Rüc96]. Im Fall der ereignisbasierten Simulation entsprechen die Ereignisse den Kunden und die Bediener den Prozessen, welche die Ereignisse bearbeiten. Die Bearbeitungsreihenfolge bei mehreren Ereignissen hängt von den Zeitstempeln dieser Ereignisse ab, die den Eintrittszeitpunkt des Ereignisses wiedergeben. Ein solches mit einem Zeitstempel behaftetes Ereignis wird durch eine Ereignisbotschaft (event notice) repräsentiert. Diese Botschaft wird entsprechend dem Zeitstempel in eine Ereigniswarteschlange (event queue) eingefügt. Eine Ablaufsteuerung (event scheduling) bearbeitet diese Warteschlange. Dabei benötigt sie Ressourcen, die den Bedienern der Warteschlangentheorie entsprechen und auf Ereignisse reagieren, indem sie gegebenenfalls neue Ereignisse generieren. Zusätzlich zu diesen Aktivitäten der Ressourcen ergeben sich Verzögerungen bei der Bearbeitung der Ereignisse bzw. bei der Bedienung der Kunden. In einem solchen Fall werden die Ereignisbotschaften in die Warteschlange der Ressource eingefügt. Die Ereignisse bilden also den Beginn oder den Abschluß der Aktivitäten und Verzögerungen.

Der Algorithmus der ereignisorientierten Ablaufsteuerung bearbeitet seine Ereigniswarteschlange, welche die chronologisch sortierten Ereignisbotschaften enthält, nach dem folgenden Schema [Cib99].

1. Initialisierung aller Ressourcen und Erzeugen von mindestens einem initialen Ereignis.
2. Entnehmen des nächsten, chronologisch ältesten Ereignisses aus der Warteschlange.
3. Fortschreiben der Simulationszeit auf die Auftrittszeit des entnommenen Ereignisses.
4. Auslösen einer Aktivität aufgrund des Ereignisses.
5. Simulation in Abhängigkeit von der Abbruchbedingung bei 2 fortsetzen oder beenden.

Die Abbruchbedingung kann der Auftritt eines Ereignisses oder das Erreichen einer bestimmten Systemzeit sein. Die ausgelöste Aktivität kann unterschiedliche Aktionen ausführen:

- Veränderung von Systemvariablen.
- Ausführung bedingter Folgeereignisse aufgrund erfüllter Bedingungen.
- Erzeugung von zukünftigen Ereignissen.

Zur Bearbeitung der Ereigniswarteschlange sind im wesentlichen Operationen zum Einfügen einer Ereignisbotschaft in die Warteschlange entsprechend der Zeit und zum Entfernen von Ereignissen in chronologisch aufsteigender Reihenfolge notwendig.

Für eine parallele Simulation muß je nach verwendetem Synchronisationsverfahren noch die Vermischung mehrerer Warteschlangen möglich sein. Zur Realisierung der Warteschlange lassen sich verschiedene Datenstrukturen verwenden [Cib99], wobei im Rahmen dieser Arbeit die lineare Liste zum Einsatz kommt.

Die diskrete ereignisgesteuerte Simulation wird für pulscodierte neuronale Netze adaptiert und genutzt [Wat94][GA98]. Die Spikes der Neuronen lassen sich als zeitbehaftete Ereignisbotschaften, die Algorithmen zur Berechnung der Neuronenaktivitäten und Synapsengewichte als Aktivitäten oder Bediener auffassen. Um eine solche Simulation zu ermöglichen, werden nachfolgend die den Algorithmen zugrundeliegenden Simulationsgleichungen aus dem Eckhorn Modell abgeleitet und darauf aufbauend wird ein Verfahren auf Basis von Ereigniswarteschlangen vorgestellt.

3.2 Simulationsgerechte Umsetzung des Modellneurons

Um das Eckhorn Modellneuron aus Kap. 2.3.3 digital simulieren zu können, muß es in eine zeit- und wertediskrete Form überführt werden. Zusätzlich sind dazu der Umfang und die Möglichkeiten des Modellneurons einzuschränken (siehe Abb. 3-1), um eine Abbildung auf digitale Simulationshardware zu ermöglichen.

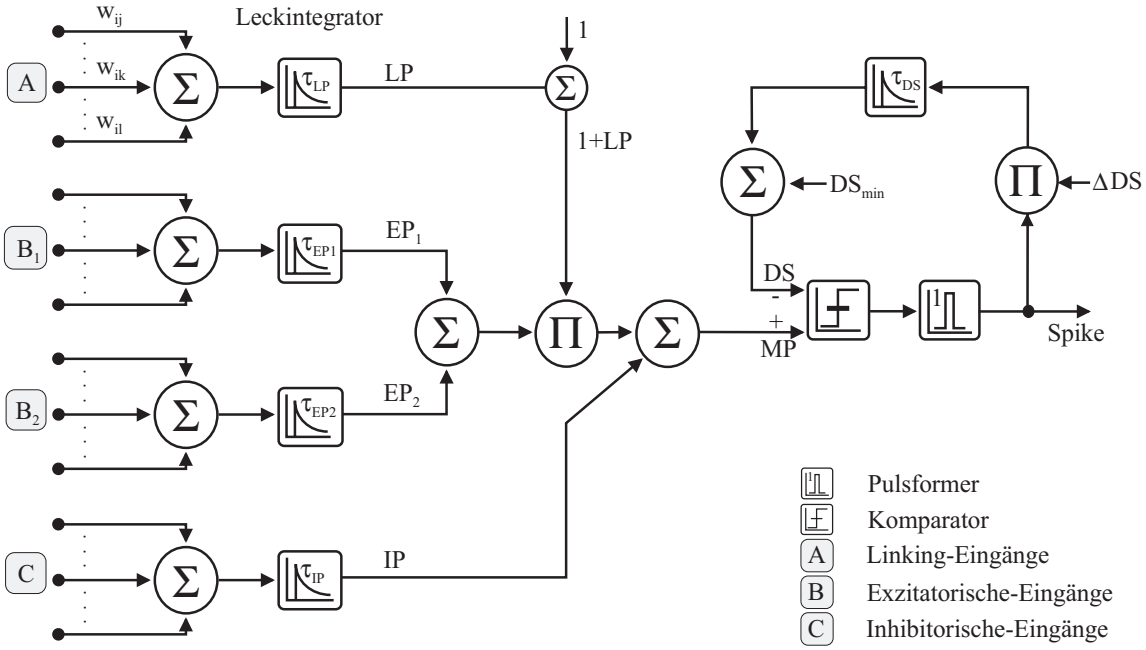

Abb. 3-1 Aufbau des zu simulierenden Modellneurons [Fra97]

Im Rahmen dieser Arbeit ist insbesondere die Anzahl der Dendritenbäume und ihre Funktion genauer festgelegt worden, als dies beim ursprünglichen Eckhorn Neuron der Fall ist. Die Festlegungen basieren auf dem für den *SPIKE128k* benutzten Modellneuron [Fra97], da die vorliegende Arbeit aus dieser Neurocomputerentwicklung hervorgegangen ist. Entsprechend

den Angaben in Kap. 2.3.3 verfügt das Modellneuron über zwei exzitatorische Dendritenbäume mit den Dendritenpotentialen EP_1 und EP_2, einen modulatorischen Linkingdendritenbaum mit dem Teilpotential *LP* und einen weiteren subtraktiven Dendritenbaum mit dem Dendritenpotential *IP*.

Zur Zeitdiskretisierung des Modells werden aus den Lösungen der Differentialgleichungen für die Dendritenpotentiale (Kap. 2.3.3) die Lösungen für die diskreten Simulationszeitpunkte $t=nT$ mit $n \in [0 \ldots N]$ bestimmt, wobei *T* die Zeitschrittlänge und *N* die Anzahl der zu simulierenden Zeitschritte ist. Diese Umwandlung soll anhand der Gleichung für das Linkingpotential durchgeführt werden, sie erfolgt in gleicher Weise für die anderen Dendritenpotentiale [Ibe99]. Um deutlich zu machen, daß es sich um Dendritenpotentiale des zeitdiskreten Modells handelt, werden die Werte nicht mehr mit $a_{LP,\,i}(t)$, also zeitkontinuierlich, sondern mit $\tilde{LP}_i(nT)$ bezeichnet. Entsprechend (Gl. 2.15) ergibt sich dann für den Aktivierungszustand des Linkingpotentials des Neurons *i* zum Zeitpunkt *nT*.

$$\begin{aligned} \tilde{LP}_i(nT) &= a_{i,\,LP}(t = nT) \\ \tilde{LP}_i(nT) &= net_{i,\,LP}(nT) * \frac{1}{\tau_{LP}} \cdot e^{-\frac{nT}{\tau_{LP}}} \cdot \sigma(nT) \end{aligned} \qquad \text{(Gl. 3.1)}$$

In einer zeitdiskreten Simulation können die Spikes nur zu den Zeitpunkten *nT* auftreten, so daß sich die Netzeingabe zu

$$net_{i,\,LP}(nT) = \sum_j w_{ij} \cdot o_j(nT) \qquad \text{(Gl. 3.2)}$$

ergibt. Die Werte der Faltung aus (Gl. 3.1) für $\tilde{LP}_i(nT)$ lassen sich mit den Herleitungen in Anhang A in Form einer diskreten Faltungssumme

$$\tilde{LP}_i(nT) \approx T \cdot \sum_{\kappa\,=\,0}^{n} net_{i,\,LP}(\kappa T) \cdot \frac{1}{\tau_{LP}} \cdot e^{-\frac{(n-\kappa)T}{\tau_{LP}}} \qquad \text{(Gl. 3.3)}$$

annähern. Aus dieser Faltungssumme kann durch einige Umformungen eine Vorschrift zur einfachen iterativen Berechnung der jeweils aktuellen Werte für $\tilde{LP}_i(nT)$ aus dem Zustand im vorangegangenen Zeitschritt und der Netzeingabe des Neurons erzeugt werden.

$$\tilde{LP}_i(nT) = \tilde{LP}_i((n-1)T)e^{-\frac{T}{\tau_{LP}}} + \tilde{net}_{i,\,LP}(nT) \qquad \text{(Gl. 3.4)}$$

Wird ferner berücksichtigt, daß $\tilde{LP}_i(nT)$ aus einem veränderlichen Anteil und einem Offset besteht, so ergibt sich mit der Substitution

$$\tilde{LP}_i(nT) = LP_i(nT) - LP_{i,\,min} \qquad \text{(Gl. 3.5)}$$

daraus die Berechnungsvorschrift für die Simulation des Linkingpotentials.

$$LP_i(nT) = (LP_i((n-1)T) - LP_{i,\,min}) \cdot e^{-\frac{T}{\tau_{LP}}} + LP_{i,\,min} + \tilde{net}_{i,\,LP}(nT) \qquad \text{(Gl. 3.6)}$$

Die Gesamtaktivierung des Neurons, die dem Membranpotential entspricht, läßt sich dann als zeitdiskretes $MP_i(nT)$ aus (Gl. 2.11) zu

$$MP_i(nT) = (EP_{1,\,i}(nT) + EP_{2,\,i}(nT)) \cdot LP_i(nT) - IP_i(nT) \qquad \text{(Gl. 3.7)}$$

ermitteln. Der in Abb. 3-1 dargestellte Offset von 1 für $LP_i(nT)$ ist dabei durch den Term $LP_{i,min}$ abgedeckt. Das Membranpotential wird wiederum dem Vergleich mit der dynamischen Schwelle zugeführt, die im zeitdiskreten Fall als $DS_i(nT)$ dargestellt wird und sich analog zu den Dendritenpotentialen mit $\Delta\tilde{DS}_i = \Delta\theta_i$ aus der Faltung (Gl. 2.23) ergibt.

$$\begin{aligned} \tilde{DS}_i(nT) &= \theta_i(t = nT) \\ \tilde{DS}_i(nT) &= \Delta\tilde{DS}_i \cdot \frac{1}{\tau_{DS}} \cdot e^{-\frac{nT}{\tau_{DS}}} \cdot \sigma(nT) * o_i(nT) \end{aligned} \qquad \text{(Gl. 3.8)}$$

Aus dieser Darstellung ergibt sich über die diskrete Faltungssumme wiederum (siehe Anhang A) eine Vorschrift zur Ermittlung der Werte für $DS_i(nT)$ aus dem Vorgängerwert und der Spikeausgabe des Neurons *i*.

$$DS_i(nT) = (DS_i((n-1)T) - DS_{i,\,min}) \cdot e^{-\frac{T}{\tau_{DS}}} + DS_{i,\,min} + \Delta DS_i \cdot o_i(nT) \qquad \text{(Gl. 3.9)}$$

Die zeitdiskrete Darstellung der Ausgabefunktion $o_i(nT)$ erfolgt durch den Vergleich des Membranpotentials mit der dynamischen Schwelle.

$$o_i(nT) = \begin{cases} 1, \text{ für } MP_i((n-1)T) \geq DS_i((n-1)T) \\ 0, \qquad\qquad \text{sonst} \end{cases} \qquad \text{(Gl. 3.10)}$$

Mit diesem Satz von Gleichungen ist das Verhalten des Neurons für eine digitale, zeitdiskrete Simulation beschrieben. Der Zustand eines solchen Neurons läßt sich damit vollständig durch die Werte seiner Dendritenpotentiale und seiner dynamischen Schwelle beschreiben. Aus diesen Werten wird ermittelt, ob das Neuron im aktuellen Zeitschritt einen Spike emit-

tiert. Die Simulation von PCNN läßt sich durch die dauernde Aktualisierung dieser fünf Werte EP_1, EP_2, IP, LP und DS realisieren, wobei die Werte in digitalen Speichern abgelegt werden können.

Die sequentielle Simulation von mehreren gekoppelten Neuronen mittels dieser Vorschriften wirft jedoch das Problem auf, daß die Netzeingabe eines Neurons nicht aufgrund von Spikes berechnet werden kann, die möglicherweise noch nicht erzeugt sind, da aktuelle Werte der präsynaptischen Neuronen noch nicht verfügbar sind. Dieses Problem läßt sich lösen, wenn neben den dendritischen Potentialen und der dynamischen Schwelle auch die Spikes eines Zeitschritts gesammelt und zwischengespeichert werden. Die präsynaptischen Spikes stehen dann im darauffolgenden Zeitschritt zur Berechnung der Netzeingabe der Neuronen zur Verfügung (Gl. 3.2). Dieser Sachverhalt wird in der Simulationsgleichung (Gl. 3.10) dadurch ausgedrückt, daß zur Ermittlung von $o_i(nT)$ das Membranpotential und die dynamische Schwelle des vorangegangenen Zeitschritts herangezogen werden. Es sei angemerkt, daß dazu nicht etwa die Werte für das Membranpotential und die dynamische Schwelle zwischengespeichert werden, sondern die durch $o_i(nT)$ repräsentierten Spikes. Der Grund für die Speicherung der Spikes und die in (Gl. 3.10) gewählte Notation wird im weiteren noch dargestellt.

Zu beachten ist ferner, daß die vollständige Netzeingabe eines Neurons *i* berechnet werden muß, bevor sein Membranpotential ermittelt werden und das Abklingen der dendritischen Potentiale und der dynamischen Schwelle erfolgen kann. Da jedoch die präsynaptischen Spikes zwischengespeichert werden und diese zur Modifikation mehrerer Netzeingaben führen, ist es sinnvoll, eine Zweiteilung eines Simulationszeitschritts vorzunehmen und in einer ersten Phase, der *Erregungsphase*, zunächst die Modifikation aller Dendritenpotentiale aufgrund der durch die Spikes erzeugten Netzeingabe zu berechnen. Nachdem dann alle Neuronen erregt wurden und die präsynaptischen Spikes des vorangegangenen Zeitschritts abgearbeitet sind, erfolgt eine weitere Phase des Zeitschritts, die *Abklingphase*, in der das Membranpotential der Neuronen berechnet wird und die Dendritenpotentiale modifiziert werden [Fra97]. Realisierungstechnische Aspekte haben beim *SPIKE128k*-System dazu geführt, daß ein Zeitschritt mit einer Abklingphase beginnt und dann erst die Erregungsphase durchgeführt wird. Da es sich hierbei nur um eine Verschiebung der Zeitschrittgrenzen handelt, die keine Auswirkung auf das Spikeverhalten der Neuronen und damit auf das Simulationsergebnis hat, wird dieser Aufbau des Zeitschritts auch in dieser Arbeit beibehalten.

Aus diesen Überlegungen ergibt sich damit ein sequentieller Simulationsalgorithmus für PCNN, der noch einmal zusammenfassend dargestellt werden soll. Die Simulation erfolgt in Zeitschritten. Ein Zeitschritt ist wiederum in eine Abklingphase und eine Erregungsphase aufgeteilt. In der Abklingphase wird aus den gespeicherten dendritischen Potentialen zunächst das Membranpotential berechnet. Danach werden die Potentiale modifiziert (*abgeklungen* [Fra97]) und gespeichert. Das Membranpotential wird mit der dynamischen

Schwelle verglichen und im Falle der Überschwelligkeit wird ein Spike erzeugt, der ebenfalls gespeichert wird. Die dynamische Schwelle wird abgeklungen, im Falle eines Spikes inkrementiert und danach gespeichert. Nachdem die Abklingphase für alle Neuronen beendet ist, erfolgt die Durchführung der Erregungsphase. In dieser Phase wird auf Basis der gespeicherten Spikes die Netzeingabe der Neuronen berechnet und die dendritischen Potentiale dieser Neuronen werden mit der Netzeingabe inkrementiert und gespeichert. Nach der Erregungsphase wird ein neuer Zeitschritt mit einer Abklingphase begonnen.

Die Abfolge der Berechnung der Simulationsgleichungen stellt sich an diesen Algorithmus angepaßt wie folgt dar. Zunächst erfolgt zu Beginn der **Abklingphase** die Berechnung des Membranpotentials aus den gespeicherten Werten des vorherigen Zeitschritts.

$$MP_i((n-1)T) = (EP_{1,i}((n-1)T) + EP_{2,i}((n-1)T)) \cdot LP_i((n-1)T) - IP_i((n-1)T) \quad \text{(Gl. 3.11)}$$

Daran schließt sich vor dem Zurückschreiben die Aktualisierung der Dendritenpotentiale an:

$$LP_i(nT) = (LP_i((n-1)T) - LP_{i,min}) \cdot e^{-\frac{T}{\tau_{LP}}} + LP_{i,min} \quad \text{(Gl. 3.12)}$$

$$EP_{1,i}(nT) = (EP_{1,i}((n-1)T) - EP_{1,i,min}) \cdot e^{-\frac{T}{\tau_{EP1}}} + EP_{1,i,min} \quad \text{(Gl. 3.13)}$$

$$EP_{2,i}(nT) = (EP_{2,i}((n-1)T) - EP_{2,i,min}) \cdot e^{-\frac{T}{\tau_{EP2}}} + EP_{2,i,min} \quad \text{(Gl. 3.14)}$$

$$IP_i(nT) = (IP_i((n-1)T) - IP_{i,min}) \cdot e^{-\frac{T}{\tau_{IP}}} + IP_{i,min} \quad \text{(Gl. 3.15)}$$

Danach erfolgt der Vergleich des Membranpotentials mit der dynamischen Schwelle und gegebenenfalls die Erzeugung und Speicherung eines Spikes.

$$o_i(nT) = \begin{cases} 1, \text{ für } MP_i((n-1)T) \geq DS_i((n-1)T) \\ 0, \text{ sonst} \end{cases} \quad \text{(Gl. 3.16)}$$

Auf Basis der Spikeberechnung erfolgt nun die Modifikation der dynamischen Schwelle, indem sie im Falle der Spikeausgabe um den Wert ΔDS_i inkrementiert wird.

$$DS_i(nT) = (DS_i((n-1)T) - DS_{i,min}) \cdot e^{-\frac{T}{\tau_{DS}}} + DS_{i,min} + \Delta DS_i \cdot o_i(nT) \quad \text{(Gl. 3.17)}$$

Nach diesen Berechnungen ist die Bearbeitung der potentiell Spike-erzeugenden Neuronen in der Abklingphase beendet. Die Berechnungen der **Erregungsphase** beschäftigen sich mit den Spike-empfangenden Neuronen. Dazu wird die Netzeingabe der Neuronen direkt auf das entsprechende Dendritenpotential addiert. Zu diesem Zweck werden zu den in der Abklingphase berechneten und gesammelten feuernden präsynaptischen Neuronen über die Netztopologie, d.h. die Verbindungsstruktur, die entsprechenden postsynaptischen Neuronen und die zugehörigen Verbindungsgewichte ermittelt. Die modifizierten dendritischen Potentiale ergeben sich damit zu:

$$LP'_i(nT) = LP_i(nT) + \sum_j \tilde{w}_{ij} o_j(nT) \qquad \text{(Gl. 3.18)}$$

$$EP'_{1,i}(nT) = EP_{1,i}(nT) + \sum_k \tilde{w}_{ik} o_k(nT) \qquad \text{(Gl. 3.19)}$$

$$EP'_{2,i}(nT) = EP_{2,i}(nT) + \sum_l \tilde{w}_{il} o_l(nT) \qquad \text{(Gl. 3.20)}$$

$$IP_i'(nT) = IP_i(nT) + \sum_m \tilde{w}_{im} o_m(nT) \qquad \text{(Gl. 3.21)}$$

Nach diesen Berechnungen sind die Neuronen aktualisiert, so daß mit den Berechnungen für den nächsten Zeitschritt begonnen werden kann.

Ein weiterer Aspekt der digitalen Simulation betrifft die Diskretisierung der Werte, womit im Fall der PCNN im wesentlichen die dendritischen Potentiale, die dynamische Schwelle, die Verbindungsgewichte und das Inkrement der dynamischen Schwelle bezüglich ihrer Bitgenauigkeiten festzulegen sind. Eine solche Festlegung hat im Rahmen des *SPIKE128k*-Projektes auf Basis umfangreicher Simulationen stattgefunden [Möl95]. Die in diesem Zusammenhang festgelegten Genauigkeiten bilden die Grundlage der im Rahmen der vorliegenden Arbeit durchgeführten Simulationen. Insbesondere im Bereich der Softwaresimulationen können die Verfahren jedoch auch mit höheren Genauigkeiten simuliert werden.

Das *SPIKE128k*-System, das in diesem Kapitel noch näher erläutert wird, ist eine Spezialhardware, die auf Festkommaberechnungen beruht. Durch diesen Umstand kommen Auflösungen der Werte zustande, die von den üblichen Formaten der digitalen Datenverarbeitung abweichen. Die Festkommawerte bestehen jeweils aus einer Anzahl von Vorkomma-Bit, Nachkomma-Bit und einem optionalen Vorzeichen. Die Notation der Auflösung erfolgt in der Form *VorzeichenVorkommastellen.Nachkommastellen*. Der Ausdruck s6.4 steht in dieser Notation für einen Festkommazahl, die aus einem Vorzeichen, 6 Vorkommastellen und 4 Nachkommastellen besteht. Ein u16-Zahl besteht aus 16 Vorkommastellen ohne Nachkommastellen und Vorzeichen und deckt den Wertebereich von 0 bis 65.535 ab.

Die im Rahmen der Simulationen benutzten Zahlenformate ergeben sich in dieser Notation wie in Tabelle 3-1 dargestellt.

Variable	Auflösung
Dendritenpotentiale EP_1, EP_2 und IP	s9.5
Linkingpotential LP	s6.4
Dynamische Schwelle DS	u17
Inkrement der dynamischen Schwelle ΔDS	u13
Gewichte der exzitatorischen und inhib. Verbindungen	s3.5
Gewichte der Linkingverbindungen	s4.4
Lernschwelle (siehe spätere Erläuterungen)	u16

Tab. 3-1 Bit-Genauigkeiten des diskreten Modellneurons [Möl95][Fra97]

Die Simulation der Eckhorn Neuronen im Rahmen dieser Arbeit erfolgt also durch die zeit- und wertediskrete Simulation eines Differentialgleichungssystems. Da es sich bei den beschreibenden Gleichungen um sehr einfache lineare Differentialgleichungen handelt, bietet es sich an, die Lösung der Differentialgleichungen zu diskretisieren und zur Simulation zu verwenden. Grundsätzlich sind jedoch auch aufwendigere Modellbeschreibungen denkbar, insbesondere erscheinen andere Modellierungen der dendritischen Potentiale auch aus Simulationsgesichtspunkten vielversprechend [Thi99]. Durch einen Ersatz der Leckintegratoren, die Filter 1. Ordnung darstellen, läßt sich dann gegebenenfalls die Feinheit der zeitlichen Diskretisierung reduzieren. Darstellungen zu den Methoden der numerischen Simulation pulscodierter neuronaler Netze und zum Vergleich der Methoden mit der hier benutzten exakten Simulation lassen sich in [HMM98][RD99] finden.

Zusätzlich zum Verhalten der Neuronen erfordert die PCNN-Simulation auch die Modellierung und Berechnung der Verbindungen. Dazu wird eine Verbindung durch ein Gewicht und eine Verzögerung beschrieben. Die Verarbeitung des Gewichts erfolgt bei der Berechnung der Netzeingabe $net_i(nT)$ eines Neurons. Die Modifikation der synaptischen Gewichte wird durch das Lernen realisiert, das im Rahmen der Arbeiten [Sch00][SH99][SSW00] ausführlich dargestellt ist und daher in dieser Arbeit nur am Rande behandelt wird. Die Nachbildung der Verbindungsverzögerungen erfolgt beim Eckhorn Modellneuron durch axonale Verzögerungen, d.h. der vom präsynaptischen Neuron emittierte Spike wird verzögert, nicht der über die Topologie verteilte, vom postsynaptischen Neuron empfangene Spike. Zu diesem Zweck ist ein spezieller Impulsformer dem Ausgang des Modellneurons (Abb. 3-1) zugeordnet, der neben Verzögerungen des Spikes auch die Aussendung von Spikemustern im Fall der Überschwelligkeit des Neurons zuläßt.

Durch den Impulsformer besteht die Möglichkeit, jedem Neuron eine eigene Impulsfolge zu-

zuordnen. Im Falle der Überschwelligkeit wird dann an Stelle eines einzigen Spikes diese Spike-Folge ausgelöst. Die Art der Impulsfolge wird durch einen Impulsvektor vorgegeben, dessen Binärstellen jeweils für eine Verzögerung um einen Zeitschritt stehen. Dieser Vektor wird dann in jedem Simulationsschritt nach einer Überschwelligkeit um eine Binärstelle nach rechts geschoben. Steht in der untersten Stelle eine logische Eins, so wird ein Spike ausgelöst (siehe Abb. 3-2).

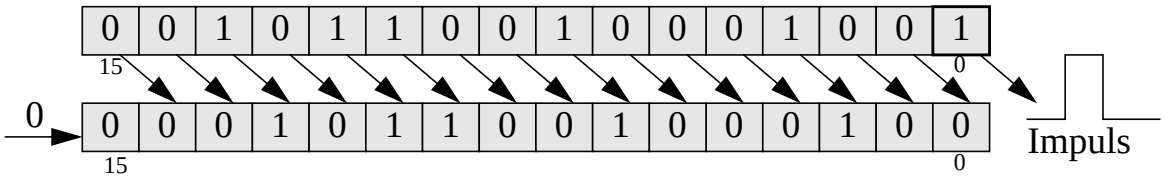

Abb. 3-2 Abarbeitung eines Impulsvektors durch den Impulsformer

Falls das entsprechende Neuron nochmals überschwellig wird bevor diese Impulsfolge abgearbeitet ist, so wird der neue Impulsvektor dem noch nicht abgearbeiteten Vektor durch eine oder-Verknüpfung überlagert. Somit werden zu jedem Zeitpunkt, an dem der geschobene alte Vektor oder der neue Vektor eine logische Eins aufweisen, Spikes generiert.

3.3 Sequentielles Zeitscheibenverfahren

Die im vorangegangen Abschnitt aufgestellten Vorschriften bilden die Grundlage einer digitalen Simulation großer PCNN, stellen aber noch kein auf einen Simulator abbildbares Verfahren dar. Um auf Basis der Vorschriften simulieren zu können, müssen Datenstrukturen und Ablaufsteuerungen konzipiert werden.

Die Umsetzung der Simulationsvorschriften auf ein einfaches Grundverfahren [FH95] [FBH95][Fra97][HFS97][FHJ99] ist Inhalt dieses Abschnitts. Dieses sequentielle Grundverfahren (siehe Abb. 3-3) ist zum einen Grundlage des *SPIKE128k*-Systems, zum anderen aber auch der Ausgangspunkt der im Rahmen dieser Arbeit konzipierten und untersuchten Simulationsmethoden. Es basiert auf der diskreten ereignisgesteuerten Simulation. Das zentrale Ereignis der Simulation ist der Spike, für den eine geeignete Ereignisbotschaft festgelegt werden muß. Ferner sind Ereigniswarteschlangen und ihre Bearbeitung zu definieren.

Da die Simulation auf einer fortlaufenden Aktualisierung der dendritischen Potentiale und der dynamischen Schwelle der Neuronen basiert, stellt dieser Satz von Neuronendaten den Ausgangspunkt des Verfahrens dar. Die Simulation wird auf diesen Daten durchgeführt, indem sie entsprechend der Vorschriften aus Kap. 3.2 gegebenenfalls in jedem Zeitschritt neu berechnet werden. Die Neuronendatensätze werden dazu in Speichern abgelegt, aus denen

sie zur Modifikation gelesen werden und in die sie nach der Berechnung zurückgeschrieben werden. Der Datensatz eines Neurons ist in diesen Speichern unter einer bestimmten Adresse zu finden, die für das Neuron charakteristisch ist. Die Adresse wird im weiteren *Neuronenadresse (NA)* genannt. Sie dient im diskreten ereignisgesteuerten Simulator als Ereignisbotschaft, d.h. bei der Emission eines Spikes durch ein Neuron wird die Neuronenadresse in der entsprechenden Ereigniswarteschlange gespeichert. Diese Warteschlange wird im weiteren *Spikeliste* genannt. Eine solche Spikeliste wird auch *spike event list* oder *address event list* genannt, die Neuronenadresse des feuernden Neurons ist entsprechend ein *address event* [Wat94]. Auf Basis dieser Datenstrukturen werden dann die Abkling- und Erregungsphasen berechnet.

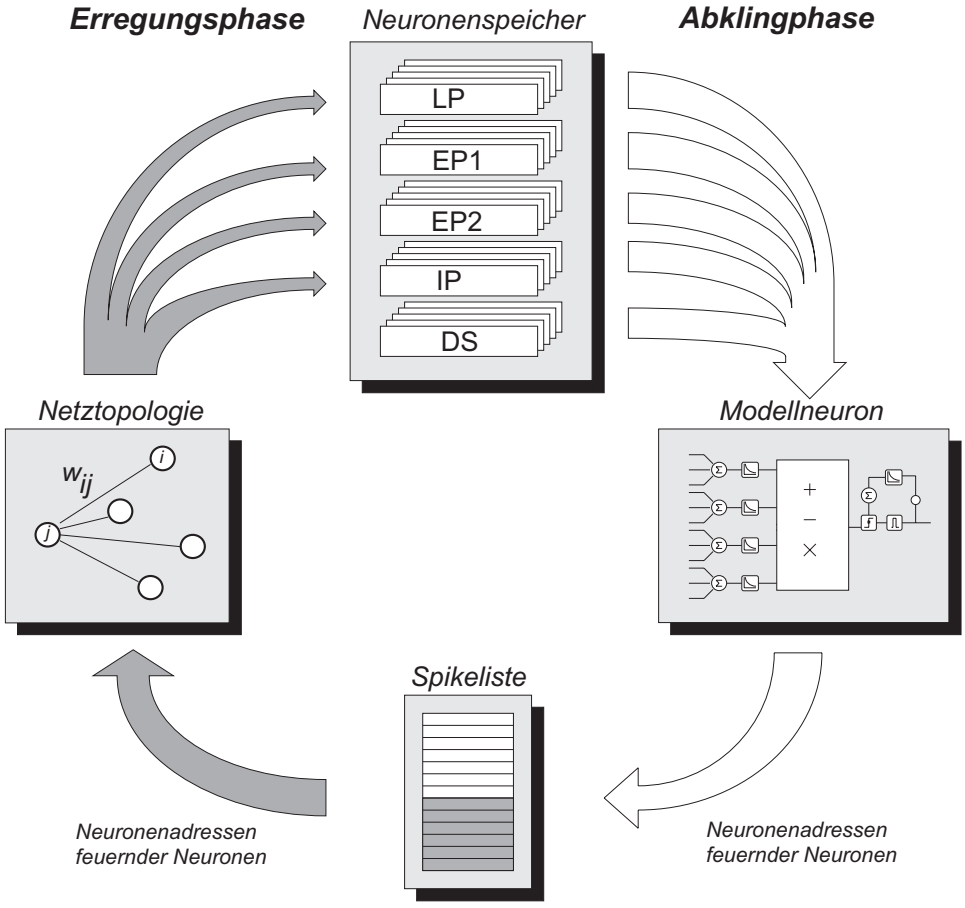

Abb. 3-3 Ablauf einer einfachen sequentiellen PCNN-Simulation

In der Abklingphase werden die Neuronenparameter aller zu berechnenden Neuronen ausgelesen und ihr Membranpotential wird berechnet. Die Parameter werden dann jeweils nach der Berechnung des einzelnen Neurons über die Leckintegratoren abgeklungen und zurückgespeichert, was möglich ist, da dieser Parametersatz in der Abklingphase nicht mehr benötigt wird. Aufgrund des berechneten Membranpotentials wird zudem festgestellt, ob das Neuron feuert. In diesem Fall wird seine Neuronenadresse in die Spikeliste eingetragen.

In der Erregungsphase wird diese Spikeliste gelesen und auf Basis der darin enthaltenen Neuronenadressen von feuernden präsynaptischen Neuronen werden über die Netztopologie, d.h. die synaptischen Verbindungen, die zu erregenden postsynaptischen Neuronen festgestellt. Deren Parameter werden dann um die Verbindungsgewichte inkrementiert. Damit sind am Ende der Erregungsphase alle zu berechnenden Neuronen abgeklungen und modifiziert, so daß der nächste Zeitschritt eingeleitet werden kann.

3.4 Methoden zur Simulationsbeschleunigung

Der vorgestellte sequentielle Grundalgorithmus bildet die Basis der durchgeführten Arbeiten. Er läßt sich unter Verwendung ereignisgesteuerter Methoden und unter Ausnutzung der in Kap. 2.4.4 beschriebenen Charakteristika großer Bildverarbeitungs-PCNN signifikant beschleunigen. Diese Optimierung des sequentiellen Verfahrens stellt neben der später vorgestellten Parallelisierung der Algorithmen den wesentlichen Ansatzpunkt zur Erreichung der gewünschten Simulationsleistung dar. Wird eine dedizierte Hardware zur Simulation verwandt, können zusätzlich Methoden des Pipelinings und der Parallelität (im weiteren noch erläutert) innerhalb des einzelnen Berechnungsschritts angewandt werden. Die einzelnen Ansätze der Simulationsbeschleunigung lassen sich grob klassifizieren [SSW00].

- *Berechnungsschritte*: Die Anzahl der Berechnungsschritte wird möglichst weitgehend reduziert, ihre Ausführung optimiert. Dazu werden spezielle Charakteristika der PCNN und Methoden der ereignisgesteuerten Simulation genutzt.
- *Speicherplatz*: Das Problem der limitierten Zahl von Speichern und der ebenso limitierten Größe dieser Speicher tritt insbesondere bei dedizierten Speichern für einzelne Berechnungseinheiten auf. Daher wird gegebenenfalls auf gemeinsam genutzte oder langsamere größere Speicher zurückgegriffen. Für die schnelleren dedizierten Speicher werden Methoden zur optimierten Nutzung angewandt.
- *Speicher-IO*: Insbesondere bei den gemeinsam genutzten größeren Speichern tritt das Problem der limitierten Bandbreite und des somit begrenzten Zugriffs auf. Daher werden Methoden zur Reduktion der Zugriffe bzw. der transferierten Datenmengen benutzt.
- *Kommunikation*: Dieses Problem betrifft in erster Linie eine parallele Simulation. Die verschiedenen Recheneinheiten müssen miteinander Daten austauschen. Die Kommunikationswege stellen eine begrenzte Ressource dar. Daher werden Methoden zur Reduktion des Kommunikationsaufkommens sowie schnelle Kommunikationsverfahren benötigt.
- *Lastbalancierung*: Dieses Problem ist typisch für parallele Systeme. Durch den Einsatz mehrerer Berechnungseinheiten ergibt sich die Frage der optimalen Ausnutzung. Dazu

werden Methoden zur Verteilung der Last auf diese Recheneinheiten benötigt.

Die letzten drei Punkte betreffen im wesentlichen die Parallelisierung der Verfahren und werden in weiteren Abschnitten der Arbeit behandelt. Zunächst sollen daher die Methoden zur Reduktion der Berechnungsschritte und zur Optimierung der Speicherung behandelt werden.

Das zeitlich dem Vorbild entsprechende Auftreten von Spikes ist das Ziel der PCNN-Simulation und daher sind Spikes die zentralen Ereignisse im Simulatorsystem. Ihre korrekte Berechnung muß gewährleistet sein und somit erfolgt eine Reduktion der Berechnungsschritte ausgehend von diesen Spikes. Ziel ist es

- möglichst nur die Neuronen zu berechnen, die einen Spike aussenden oder empfangen,
- nur die Teile des Neurons zu berechnen, die an Spikeaussendung oder Spikeempfang aktuell beteiligt sind, und
- nur die Synapsen zu berechnen, die einen Spike übertragen.

Dazu müssen ereignisorientierte Ablaufsteuerungen konzipiert werden, die Spikes in Ereigniswarteschlangen sammeln und möglichst nur in Abhängigkeit dieser Ereignisse Aktionen ausführen. Nachfolgend werden zunächst zwei Verfahren aus [Fra97] vorgestellt, die im Rahmen des *SPIKE128k* benutzt wurden, sowie anschließend zwei Verfahren [WHR99a], die darauf aufbauend im Rahmen dieser Arbeit konzipiert wurden.

3.4.1 Spikeliste und senderorientierte Verbindungslisten

Die zentrale Ereigniswarteschlange, die Spikeliste, wurde bereits im Kap. 3.3 als Bestandteil des Grundverfahrens vorgestellt. Auf Basis dieser Spikeliste läßt sich eine signifikante Beschleunigung der Erregungsphase des Zeitschritts sowie eine Reduktion des Speicherbedarfs erreichen. Dazu bedarf es einer besonderen Art der Speicherung der Verbindungsstruktur des neuronalen Netzes. Üblicherweise wird eine solche Struktur in Form einer Gewichtsmatrix gespeichert [KS96]. Der Spaltenindex dieser Matrix entspricht dabei der Neuronennummer des präsynaptischen Neurons, der Zeilenindex der des postsynaptischen Neurons. Der Matrixeintrag an der entsprechenden Position bestimmt das Gewicht der Verbindung. Mit Hilfe dieser Darstellung lassen sich beliebig verknüpfte Strukturen bis hin zu einem vollständig verbundenen Netz notieren.

Der große Nachteil der Matrixnotation liegt in der Tatsache, daß die Datenmenge quadratisch mit der Anzahl der Neuronen wächst. Dieses Problem wird im Falle der sehr großen Bildverarbeitungs-PCNN schnell zum limitierenden Faktor der Simulation. Für das in Kap. 2.4.1 beschriebene Weitzelnetz mit 118.801 Neuronen wäre eine Matrix mit mehr als 14 Milliarden Einträgen nötig. Tatsächlich verfügt das Netz aber nur über etwa 4 Millionen Verbin-

dungen, d.h. der überwiegende Teil der Matrixelemente würde den Wert 0 aufweisen. Die typischerweise spärliche Verknüpfung von Bildverarbeitungs-PCNN läßt also die Matrixnotation der Verbindungsstruktur ineffizient erscheinen. Eine dem Problem angepaßte Notation bildet die Verwendung von Verbindungslisten, in denen jeweils die Verbindungen eines Neurons aufeinanderfolgend abgelegt werden. Aufgrund der Unidirektionalität der Synapsen ist zwischen Verbindungen zu unterscheiden, die zum Neuron führen oder solchen die vom Neuron wegführen. Bezüglich der zum Neuron führenden Verbindungen ist das Neuron postsynaptisch, d.h. es ist der Empfänger der übertragenen Spikes. Die vom Neuron wegführenden Verbindungen repräsentieren die Synapsen am Axon des Neurons, d.h. das Neuron ist bezüglich dieser Verbindungen präsynaptisch, es sendet über diese Verbindungen. Die Aufnahme beider Verbindungstypen in die Verbindungsliste würde zu einer doppelten Speicherung der Verbindungen führen. Im Rahmen des in Kap. 3.3 vorgestellten Grundverfahrens werden jedoch zunächst nur die Verbindungen benötigt, über welche die in der Spikeliste gespeicherten präsynaptischen Spikes Erregungen an die postsynaptischen Neuronen senden. Eine Speicherung nur solcher Verbindungen führt zu einer *senderorientierten* Darstellung. Erlaubt der Simulator zudem eine im Rahmen des gesamten Speichers flexible Größe der Listen, so wird diese in Abb. 3-4 abgebildete Speicherung als *dynamische senderorientierte Speicherung* der Netztopologie bezeichnet [Fra97].

Da sich bei der Verwendung solcher Verbindungslisten die Neuronenadresse des postsynaptischen Zielneurons nicht aus der Position des Eintrags in einer Matrix ergibt, muß diese Adresse zusammen mit dem Verbindungsgewicht und einigen weiteren Parametern gespeichert werden. Ein solches Verbindungselement wird im weiteren *Erregungsinformation* genannt, die Liste, die diese Informationen für ein präsynaptisches Neuron enthält heißt dementsprechend *Erregungs-Informations-Block* (EIB). Die EIBs sind aufeinanderfolgend im Topologiespeicher abgelegt und werden indirekt adressiert.

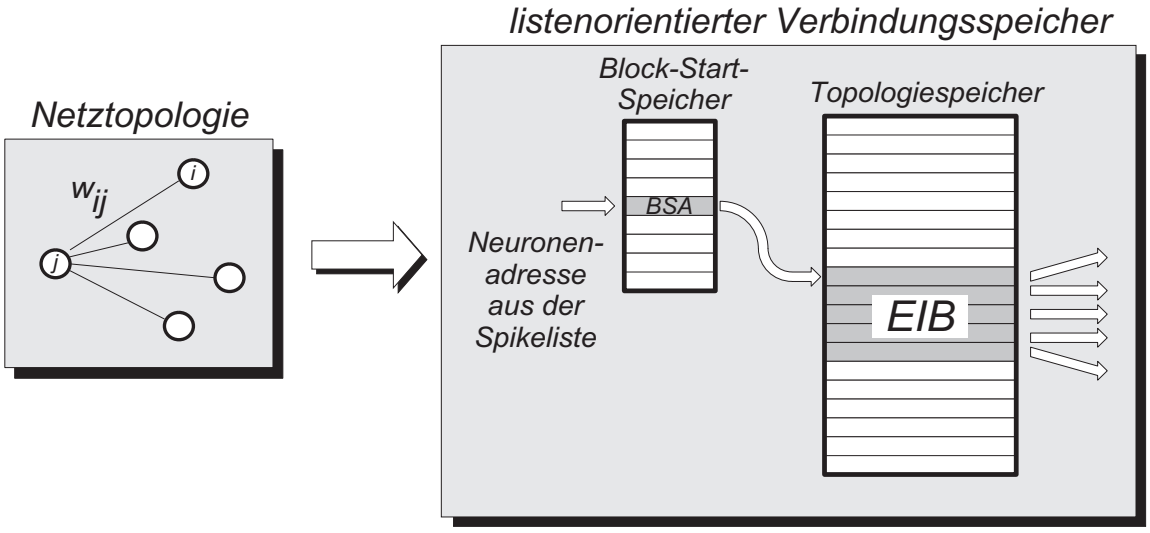

Abb. 3-4 Dynamische senderorientierte Speicherung der Netztopologie

Zu jedem EIB wird eine *Block-Start-Adresse* (BSA), die auf den Anfang des Blocks im Topologiespeicher zeigt, in einem *Block-Start-Speicher* (BSS) abgelegt. Diese BSA wird im BSS wiederum mittels der Neuronenadresse des feuernden präsynaptischen Neurons adressiert. Über die BSA läßt sich somit der EIB mit allen Verbindungen eines Neurons zu seinen postsynaptischen Nachfolgern adressieren.

Durch die flexible Größe der EIBs können im Rahmen der Gesamtgröße des Topologiespeichers beliebige Verknüpfungsstrukturen abgelegt werden. Zusätzlich zu dieser kompakten Topologierepräsentation bietet die senderorientierte Speicherung jedoch noch einen weiteren wesentlichen Vorteil. Da in der Erregungsphase nur die Adressen feuernder präsynaptischer Neuronen aus der Spikeliste verwendet werden und in den von diesen Neuronen adressierten EIBs alle Verbindungen dieses speziellen Neurons abgelegt sind, erfolgt implizit nur die Berechnung der tatsächlich Spike-übertragenden Synapsen. Eine solche Beschränkung der Berechnungsschritte ist eine der Hauptforderungen an eine beschleunigte Simulation.

3.4.2 Abklingliste

Nachdem durch die Einführung der beschriebenen einfachen senderorientierten Topologieverarbeitung die Erregungsphase weitgehend optimiert ist, sind im *SPIKE128k*-System Ansätze zur Beschleunigung der Abklingphase implementiert worden [Fra97]. Der einfache Grundalgorithmus sieht vor, daß in dieser Phase alle Neuronen nacheinander abgearbeitet werden, indem ihre Neuronendaten gelesen werden, ihr Membranpotential berechnet wird und die dendritischen Teilpotentiale sowie die Schwelle aktualisiert und geschrieben werden. Charakteristisch für Bildverarbeitungs-PCNN (siehe Kap. 2.4) ist jedoch, daß nur eine geringe Anzahl von Neuronen am Simulationsgeschehen beteiligt ist. Alle anderen Neuronen befinden sich auf Ruhepotentialen und können ohne eine externe Anregung kein Aktionspotential erzeugen, d.h. sie sind am Spikeverhalten des Netzes unbeteiligt. Eine Berechnung dieser Neuronen in der Abklingphase ist daher überflüssig. Um nur die aktiven, potentiell Spike-erzeugenden Neuronen zu bearbeiten, werden die Neuronenadressen dieser Neuronen in einer weiteren Ereignisliste, der *Abklingliste*, gespeichert. Diese Liste wird jeweils während einer Abklingphase für den darauffolgenden Zeitschritt neu erzeugt.

Dazu wird zunächst in der Abklingphase des Zeitschritts die Abklingliste des vorherigen Zeitschritts gelesen. Die enthaltenen Neuronen werden aktualisiert und ihr Membranpotential wird berechnet. Gegebenenfalls erfolgt der Eintrag der Neuronenadresse in die Spikeliste. Zusätzlich werden die aktualisierten Werte mit den Ruhewerten verglichen. Nur wenn die Werte sich von den Ruhewerten unterscheiden, wird die Neuronenadresse erneut in die Abklingliste eingetragen. Da nur die aus der Abklingliste gelesenen Neuronen auch wieder geschrieben werden können, ist die Anzahl der Einträge in die neue Abklingliste kleiner oder gleich der Anzahl der Einträge in der alten Liste. Daher kann diese alte Liste während der

Abklingphase überschrieben werden, ohne noch nicht bearbeitete Neuronenadressen zu löschen. Zu beachten ist, daß auch Neuronen, deren Impulsfolge (siehe Kap. 3.2) nicht vollständig abgearbeitet ist, wieder in die Abklingliste eingetragen werden. Somit entsteht auf Basis eines einzigen Listenspeichers während der Abklingphase ein Teil der neuen Abklingliste.

In diese Liste müssen während der Erregungsphase die Neuronenadressen der erregten Neuronen eingetragen werden, soweit diese noch nicht in der Liste vorhanden sind. Neu erregte Neuronen weisen dendritische Potentiale ungleich den Ruhewerten auf und müssen in der nächsten Abklingphase neu berechnet werden. Um zu vermeiden, daß mehrfach erregte Neuronen mehrfach in die Liste eingetragen werden oder solche Neuronen, die sich ohnehin noch in der Abklingliste befinden, zusätzlich aufgenommen werden, wird ein Markierungs-(Tag)-Speicher verwendet, in dem unter der Neuronenadresse ein Tag-Bit gesetzt wird, wenn sich das Neuron in der Abklingliste befindet. Beim Lesen der Liste wird dieses Bit zurückgesetzt. Nach der Erregungsphase steht dann für den nächsten Zeitschritt eine vollständige Abklingliste zur Bearbeitung bereit. Zu Beginn der Simulation wird die Liste einmalig mit allen vorhandenen Neuronenadressen gefüllt.

Die Abklingliste bildet damit den Ausgangspunkt zur Steuerung der Abklingphase, d.h. die Berechnungen der Abklingphase werden durch die in der Liste enthaltenen Ereignisbotschaften in Form von Neuronenadressen angestoßen. Für die Erregungsphase wird diese Aufgabe durch die Spikeliste erfüllt. Durch die Steuerung der Simulation auf Basis dieser Ereignislisten wird die Simulationszeit an die Netzaktivität gekoppelt, die neben dem Netzaufbau auch von der Form des Eingangsstimulus abhängt. Eine Abschätzung der Leistungsfähigkeit des Simulationsverfahrens kann daher nur auf Basis von Beispielen bzw. auf Basis der aus diesen Beispielen gewonnenen statistischen Daten erfolgen.

3.4.3 Wiedervorlageliste und Topologie-Cache

Bei der Spikeliste und der Abklingliste handelt es sich um Ereigniswarteschlangen, die nur Ereignisse aus dem vorangegangenen Zeitschritt enthalten. Damit entfällt die Notwendigkeit für einen Zeitstempel oder eine chronologische Organisation der Daten. Im Fall der Abklingliste führt dieses jedoch zu unnötigem Berechnungsaufwand. Wenn nämlich ein Neuron aus der Abklingliste gefeuert hat, so befindet es sich danach für eine Anzahl von Zeitschritten in der Refraktärperiode und kann nicht erneut feuern. Erhält dieses Neuron zudem keine neuen Anregungen durch andere Neuronen, so werden seine Potential- und Schwellwerte während der Abklingphasen ausschließlich in kleinen Schritten abgeklungen. Dieser Vorgang erfolgt gegebenenfalls bis zum Erreichen der Ruhewerte und hat keinen weiteren Einfluß auf das Spike-Verhalten des gesamten Netzes. Daher können die Abklingschritte zu größeren Änderungen zusammengefaßt werden, die immer dann erfolgen, wenn ein aktueller Zustand des Neurons benötigt wird. In der Zwischenzeit kann das Neuron in ei-

ner Ereignisliste abgelegt werden. Wenn das Neuron potentiell wieder feuern kann oder aber eine neue Eingangserregung erhält, so muß es *wiedervorgelegt* werden. Die verwendete, mit Zeitstempeln versehene Ereigniswarteschlange wird im weiteren daher *Wiedervorlageliste* genannt [WHR99a]. Für diese Wiedervorlage wird nach dem Feuern eines Neurons die Zeit berechnet, die benötigt wird, um die inkrementierte dynamische Schwelle in den Bereich des Membranpotentials abzuklingen. Für diese *Wiedervorlagezeit* wird das Neuron dann abgelegt. Einzig im Fall einer Erregung durch ein anderes Neuron erfolgt eine vorzeitige Wiedervorlage.

Der Einsatz der Wiedervorlageliste ermöglicht eine zumindest teilweise Entkopplung des Simulationsaufwandes von der Feinheit der zeitlichen Diskretisierung. Der Simulationsaufwand $c(t, T)$ für den Zeitschritt t läßt sich dazu durch die Anzahl zu berechnender Neuronen in der Abklingphase $a(t, T)$ und die Anzahl der zu berechnenden Synapsen in der Erregungsphase darstellen, wobei T der durch einen Zeitschritt repräsentierten Zeit entspricht. Die Anzahl der zu berechnenden Synapsen hängt von der Anzahl der erzeugten Spikes und der Netztopologie ab. Der Einfluß der Netztopologie soll im weiteren annähernd durch die mittlere Konnektivität k, d.h. durch die durchschnittliche Anzahl von Verbindungen pro Neuron dargestellt werden. Mit einer Spikeanzahl $s(t, T)$ pro Zeitschritt ergibt sich dann für den Gesamtaufwand der folgende Ausdruck.

$$c(t, T) = a(t, T) + s(t, T) \cdot k \qquad \text{(Gl. 3.22)}$$

Die gesamte Spikeanzahl eines Zeitraums ist prinzipiell unabhängig von der Feinheit der zeitlichen Diskretisierung, d.h. bei einer genügend genauen Modellierung entspricht sie dem biologischen Vorbild. Wird eine Zeiteinheit des kontinuierlichen Verlaufs in doppelt so viele Zeitschritte wie zuvor eingeteilt, so wird sich die Spikeanzahl pro Zeitschritt halbieren. Damit ist der Aufwand, den die Erregungsphase am Gesamtaufwand für die Simulation einer vorgegebenen Zeitspanne des kontinuierlichen Vorbildes ausmacht, von der Feinheit der Zeitdiskretisierung unabhängig. Im Gegensatz dazu bleibt bei der Verwendung der Abklingliste der Aufwand für eine Abklingphase im Falle einer Verkürzung der Zeitschrittlänge T nahezu konstant. Der Abklingvorgang der Neuronen wird dann nur in kleinere Schritte aufgeteilt, so daß die Anzahl der benötigten Abklingschritte annähernd linear mit der Feinheit der Zeitdiskretisierung steigt. Zusammenfassend steigt also der Simulationsaufwand bei Verwendung der Abklingliste, wenn die simulierte Zeit in mehr Zeitschritte eingeteilt wird.

Bei Verwendung der Wiedervorlageliste zur Steuerung der Abklingphase werden im Idealfall nur solche Neuronen vorgelegt, d.h. berechnet, die entweder durch das Abklingen der dynamischen Schwelle einen neuen Spike erzeugen oder aber durch einen solchen von einem anderen Neuron erregt werden. Damit ist auch der Aufwand der Abklingphase an die Spikeanzahl $s(t, T)$ gekoppelt und somit ist der Gesamtaufwand von der Feinheit der Zeitdiskretisierung unabhängig, da sich $s(t, T)$ im Mittel invers proportional zur Feinheit der

Zeitdiskretisierung bzw. proportional zu T verhält. Der Aufwand $a(t, T)$ läßt sich dann aus der Anzahl der erzeugten Spikes und der Anzahl der durch Spikes aus dem vorangegangenen Zeitschritt neu erregten Neuronen berechnen. Die Anzahl der neu erregten Neuronen hängt dabei wiederum mit der mittleren Konnektivität k zusammen. Allerdings kann es vorkommen, daß ein Neuron über mehrere Verbindungen gleichzeitig erregt wird. Damit ergibt sich für den Gesamtaufwand eines Zeitschritts

$$c(t, T) = s(t, T) + s(t-1, T) \cdot \tilde{k} + s(t, T) \cdot k, \qquad \text{(Gl. 3.23)}$$

wobei $\tilde{k}$ zur Ermittlung der Anzahl der neu erregten Neuronen kleiner oder gleich der mittleren Konnektivität k ist. Damit ist der Gesamtaufwand an die gesamte Spikeanzahl eines Zeitraums gekoppelt und von der Feinheit der Zeitdiskretisierung weitgehend unabhängig. Wird der Zeitraum in doppelt soviele Zeitschritte eingeteilt, halbiert sich die Spikeanzahl pro Zeitschritt $s(t, T)$ im Mittel.

Die Berechnung der Wiedervorlagezeit, also der Zeit, nach der eine aufgrund der Spikeemission inkrementierte dynamische Schwelle durch Abklingen das ebenfalls abklingende Membranpotential erneut trifft, ist rechenaufwendig, da sich das Membranpotential aus mehreren Teilpotentialverläufen zusammensetzt. Stattdessen kann jedoch recht einfach der Zeitpunkt berechnet werden, zu dem die abklingende dynamische Schwelle den Wert des Membranpotentials zum Ablagezeitpunkt erreicht (siehe Abb. 3-5). Nach dieser verkürzten Zeit bis zur Wiedervorlage kann dann die Berechnung erneut durchgeführt werden, so daß der Zeitpunkt des Zusammentreffens von dynamischer Schwelle und Membranpotential iterativ über mehrere Wiedervorlageschritte erreicht wird.

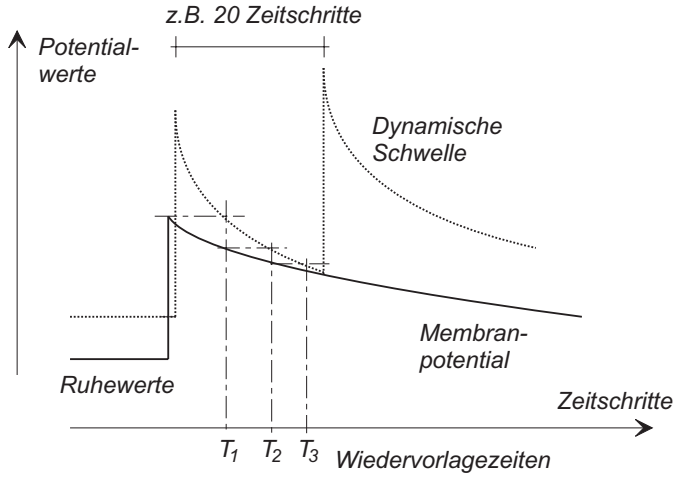

Abb. 3-5 Iterative Berechnung des Wiedervorlagezeitpunktes

Da Neuronen, die aktiv am Simulationsgeschehen teilnehmen, gewöhnlich mehr als eine Eingangserregung erfahren, ist zu erwarten, daß eine Ablage bis zum berechneten Wiedervorlagezeitpunkt ohnehin nicht erreicht wird, sondern das Neuron aufgrund dieser Erregun-

gen außerplanmäßig schon früher wiedervorgelegt werden muß. Die Berechnung des Wiedervorlagezeitpunktes T_{WV} vereinfacht sich bei der iterativen Annäherung zu einer Logarithmus-Berechnung. Mit der dynamischen Schwelle *DS* und dem Membranpotential *MP* ergibt sich damit der folgende Ausdruck, wobei τ_{DS} die Zeitkonstante des Leckintegrators der dynamischen Schwelle ist.

$$T_{WV} = \tau_{DS} \cdot (\ln(DS) - \ln(MP)) \qquad \text{(Gl. 3.24)}$$

Mit Hilfe dieser Wiedervorlagezeit läßt sich der Zeitschritt ermitteln, in dem das Neuron erneut berechnet werden muß. Mit diesem Zeitstempel versehen wird die Neuronenadresse dann in die Ereigniswarteschlange der Wiedervorlageliste eingeordnet. Zusätzlich muß noch der Zeitpunkt der Ablage gespeichert werden, da mit Hilfe dieses Wertes der durchzuführende Abklingschritt bei der Wiedervorlage berechnet wird. Mit Hilfe dieses Ablagezeitpunktes kann auch bei einer außerplanmäßigen Wiedervorlage aufgrund einer Erregung die Größe des Abklingschrittes bestimmt werden.

Für eine Implementierung bietet sich eine Begrenzung der Zeitstempel auf eine bestimmte Wortbreite an. Die Berechnung der Stempel erfolgt dann über eine Modulo-Operation mit der durch die Stempel darstellbaren Ablagezeit. Weiterhin ist es sinnvoll, einen Stempel mit der maximal darstellbaren Zeit für Neuronen vorzusehen, die sich auf Ruhewerten befinden, und einen Stempel mit dem Wert 0 für aktuell zu bearbeitende Neuronen, so daß sich eine maximale Ablagezeit von $2^n - 2$ Zeitschritten bei einer Stempelwortbreite von *n* bit ergibt. Die Neuronenadressen können zusammen mit den Zeitstempeln in einer einzigen Liste gesammelt werden. Bei der Verarbeitung werden dann jeweils die Neuronen übergangen und direkt in die neue Liste eingetragen, die keinen aktuellen Zeitstempel aufweisen. Diese Lösung bietet sich für eine Softwarerealisierung an. Im Falle einer Hardwarerealisierung erscheint es günstig, für jeden möglichen Wiedervorlagezeitpunkt (begrenzt durch die Modulo-Operation) eine eigene Liste mit einer eigenen Verwaltungseinheit vorzuhalten. Mit der jeweils aktuellen Liste kann dann eine Verarbeitungspipeline, die pro Takt ein Neuron bearbeiten kann, ohne Unterbrechung gespeist werden.

Eine weitere im Rahmen dieser Arbeit entwickelte Methode zur Optimierung des sequentiellen Simulationsverfahrens betrifft die Behandlung der Netztopologie, d.h. der in den EIBs (Erregungsinformationsblöcke) gespeicherten Verknüpfungsstruktur. Bei diesem Topologiespeicher handelt es sich um den größten Datenblock, der bei der vorgestellten PCNN-Simulation benötigt wird, da die Anzahl der Verbindungen gewöhnlich um ein bis zwei Größenordnungen über der Anzahl der Neuronen liegt (siehe Kap. 2.4.4). Damit ergibt sich das Problem des in der Bandbreite begrenzten Zugriffs auf diesen Speicher. Üblicherweise werden solche Probleme in der Rechnerarchitektur durch den Einsatz eines *Cache*-Speichers gemildert.

Ein Cache ist ein schneller Pufferspeicher, der zwischen einen langsamen Hauptspeicher und eine Recheneinheit geschaltet wird, um häufig benötigte Daten dort temporär abzulegen und der Recheneinheit einen gegenüber dem Hauptspeichertransfer schnelleren Zugriff auf diese Daten zu ermöglichen. Der Cache-Speicher ist dabei wesentlich kleiner als der Hauptspeicher [TN92]. Das Problem der Cache-Verwaltung besteht in der Auswahl der zu puffernden Daten, der Festlegung der Größe der Datenabschnitte und der Wahl eines Verfahrens, das die eingelagerten Blöcke gegebenenfalls durch andere ersetzt.

Im Bereich der Neuronale-Netze-Simulation sind zu dieser Thematik insbesondere Untersuchungen von [TN92] durchgeführt worden, die sich allerdings mit klassischen Neuronenmodellen beschäftigen. Neben einem Cache für die Neuronenparameter wird auch hier eine Pufferung der Verknüpfungsdaten betrachtet, da wie bei den PCNN die Speicherung der Verbindungsgewichte ein kritisches Element in bezug auf die Simulationsleistung ist. Untersucht wird dabei die sogenannte *hit-* oder *miss-ratio* des Caches, die das Verhältnis der Anzahl der im Cache gefundenen bzw. nicht gefundenen Verbindungen zur Anzahl der insgesamt gesuchten Verbindungen beschreibt. Die hit-ratio wird auch *Trefferrate* des Caches genannt. Zur Erreichung einer guten Trefferrate muß eine genügend hohe Lokalität der Aktivität im Netz vorausgesetzt werden, d.h. die Aktivität muß für eine längere Zeit im gleichen Netzbereich verbleiben. Wenn dann der Cache genügend viele Verbindungen aufnehmen kann und zudem benachbarte Verbindungen mit in den Cache gelegt werden, indem größere zusammenhängende Datenpakete aus den Topologiedaten gewählt werden, so ergeben sich Trefferraten von über 90%. Erreicht werden diese Raten, indem während der Verarbeitung eines Eingangsmusters auf eine Ersetzung der Datenpakete im Cache - der sogenannten *cachelines* - verzichtet wird. Dabei muß eine Cache-Größe gewählt werden, die möglichst die Einlagerung aller durch das Eingangsmuster betroffenen Verbindungen ermöglicht.

Für die PCNN-Simulation gelten diese Annahmen prinzipiell auch, jedoch sind einige weitere Randbedingungen zu beachten. Zunächst einmal müssen nicht für alle aktiven Neuronen die Verbindungen im Cache gehalten werden, sondern nur für diejenigen, die einen Spike emittieren. Solche Neuronen sind nach der Spike-Emission refraktär, d.h. sie werden für eine gewisse Zeit keinen weiteren Spike erzeugen. Allerdings ist ihre Aktivität weiterhin erhöht, so daß sie nach der Refraktärzeit mit höherer Wahrscheinlichkeit erneut feuern als andere Neuronen. Daher ist es sinnvoll, die Verbindungen dieser Neuronen im Cache zu halten. Üblicherweise in der Rechnertechnik verwendete Caching-Strategien würden gerade diese Verbindungsdaten ersetzen, da ihre letzte Nutzung am weitesten zurückliegt, und damit die Trefferrate verschlechtern, da diese Daten mit hoher Wahrscheinlichkeit als nächstes gebraucht werden. Eine für PCNN sinnvolle Caching-Strategie muß also einmal eingelagerte Verbindungsdaten möglichst lange im Cache halten, bzw. gegebenenfalls die zuletzt benutzten Verbindungen aus dem Cache entfernen. Neben Neuronen, die gefeuert haben, werden

auch Neuronen, die Erregungen durch die Spikes erfahren haben, mit erhöhter Wahrscheinlichkeit feuern. Gleiches gilt für Neuronen, deren Membranpotential sich nahe an der dynamischen Schwelle befindet. Weitere Aussagen über die Wahrscheinlichkeit eines Spikes lassen sich aus den Abklingverläufen der Neuronenparameter gewinnen, so daß ein weites Feld für das Auffinden von Cache-Algorithmen zur Verfügung steht.

Ein weiterer Punkt bei der Konzeption eines Cache-Verfahrens für PCNN ist die Auswahl einer Verwaltungsstrategie, die im Bereich der konventionellen Rechnertechnik durch die Festlegung einer geeigneten Organisation und Länge der cachelines erfolgt. Für das im Rahmen dieser Arbeit untersuchte Verfahren bietet sich eine der dynamischen senderorientierten Topologiespeicherung (siehe Kap. 3.4.1) nachempfundene Organisation an. Es werden dann immer vollständige EIBs eingelagert, da im Falle eines Spikes immer alle Verbindungen des Neurons benötigt werden, und diese EIBs werden indirekt über einen Cache-BSS adressiert. Befindet sich im Cache-BSS unter der Neuronenadresse des feuernden Neurons ein Null-Zeiger, so ist für das Neuron kein Cache-Eintrag vorhanden. Da die EIBs unterschiedliche Längen haben, würde bei einer Ersetzungsstrategie für den Cache die Notwendigkeit zum Umkopieren der anderen EIBs entstehen, wenn ein alter EIB gelöscht wird. Andernfalls würden Lücken im Speicher entstehen. Aus diesem Grund wurde im Rahmen dieser Arbeit zunächst ein sehr einfaches Verfahren implementiert, das aufgrund der im weiteren noch darzustellenden guten Ergebnisse dann beibehalten wurde.

Dazu wird der Cache in Form eines Cache-BSS und eines EIB-Speicherblocks organisiert, in den im Laufe der Simulation die EIBs feuernder Neuronen eingelagert werden. Ist der Cache gefüllt, so werden für eine feste Anzahl von Zeitschritten keine Änderungen vorgenommen. Nach dieser Wartezeit wird der Cache komplett geleert, indem die Zeiger des Cache-BSS alle auf Null zurückgesetzt werden. Der Schreibzeiger für den EIB-Datenbereich wird ebenfalls auf Null gesetzt, so daß der Cache wieder gefüllt werden kann. Dieses Verfahren ist denkbar einfach und daher mit sehr geringem Aufwand an Ressourcen, insbesondere an Rechenzeit, zu implementieren. Durch die Beibehaltung des EIBs als Dateneinheit fügt sich das Verfahren nahtlos in die gesamte Topologiedatenverwaltung ein. Da keine Ersetzung einzelner EIBs erfolgt, kann über eine geeignet gewählte Wartezeit auch bei vollem Cache weitgehend verhindert werden, daß noch benötigte EIBs vorzeitig gelöscht werden. Das Verfahren eignet sich durch seine Einfachheit auch für die im weiteren noch dargestellte Parallelisierung sowie für Hardwareimplementierungen, da nur sehr wenige Daten zur Cache-Verwaltung benötigt werden. Dieses sind die Information, daß ein Neuron feuert, sein EIB, ein Zeitzählerwert für die Zeit bis zur Löschung und der Füllstand des Caches.

Da die im Rahmen dieser Arbeit durchgeführten Simulationen gezeigt haben, daß sich mit diesem einfachen Verfahren sehr gute Ergebnisse erzielen lassen, wurde auf die Untersuchung aufwendigerer Verfahren verzichtet. Die Simulationen, die mit den bisher beschriebenen Verfahren durchgeführt wurden, sind Inhalt des folgenden Abschnitts.

3.5 Ergebnisse und Folgerungen

Zur Untersuchung der in Kap. 3.3 und Kap. 3.4 vorgestellten Verfahren wurde ein Softwaresimulator für Sun-Workstations unter dem Betriebssystem Solaris implementiert. Dieser Simulator realisiert das Simulationsverfahren des Neurocomputers SPIKE128k [Fra97], welches wiederum auf dem in Abb. 3-3 dargestellten Simulationskreislauf basiert. Durch ein modulares Design des Simulators lassen sich die in Kap. 3.4 vorgestellten Verfahren einfügen bzw. austauschen. Die Berechnungen erfolgten mit den in Kap. 3.2 aufgelisteten Genauigkeiten. Der Neurocomputer SPIKE128k läßt solche Untersuchungen nicht zu, da entscheidende Parameter nicht während der Simulation überwacht werden können.

Um die Verfahren einschätzen zu können, ist die Ermittlung verschiedener Parameter sinnvoll, die sich in Werte zum Rechenaufwand bzw. zur Rechenzeit, zum Speicheraufwand und zur Kommunikation einteilen lassen. Alle diese Werte bestehen aus einem statischen Anteil für eine einzelne Aktion oder einen einzelnen Datenblock und aus einem dynamischen Anteil, der sich aus dem simulierten Netz und seinem Verhalten ergibt. Die dynamischen Parameter ergeben sich aus der Anzahl der Neuronen, Synapsen oder Spikes, die in einer Simulationsphase zur Berechnung, Speicherung oder Kommunikation anfallen.

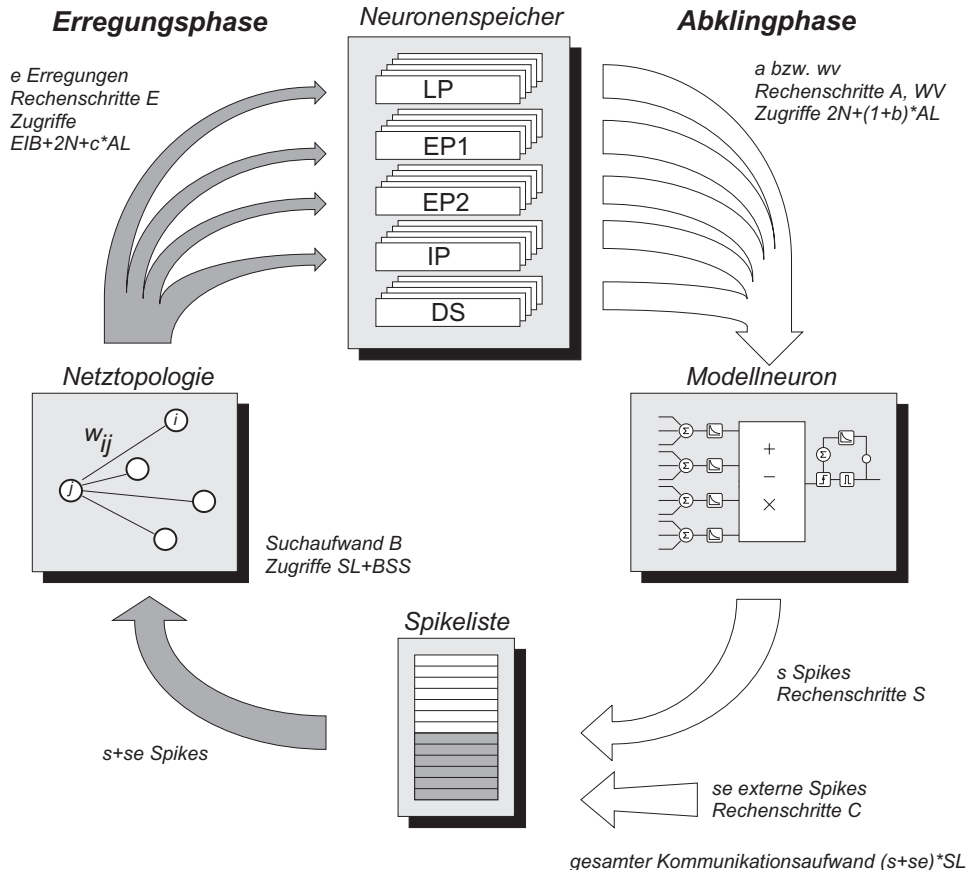

Abb. 3-6 Ermittelte Parameter der sequentiellen Simulation

Für die Einschätzung der Verfahren und zur Vorhersage der Simulationsgeschwindigkeiten auf verschiedenen Simulationsplattformen ist die Messung der folgenden dynamischen Daten des sequentiellen Simulationsverfahrens von Interesse (siehe auch Abb. 3-6).

- Der Füllstand der Abklingliste *a,* d.h. die Anzahl der Neuronen, die sich zu Beginn der Abklingphase in der Liste befinden und damit berechnet werden müssen. Wird keine Abkling- oder Wiedervorlageliste eingesetzt, so entspricht *a* der Neuronenanzahl *n* aller Neuronen.
- Die Anzahl *wv* der Neuronen in der Wiedervorlageliste, die einen aktuellen Zeitstempel haben und damit in der Abklingphase berechnet werden müssen. Diese Anzahl ersetzt bei Einsatz der Wiedervorlageliste den Wert *a*.
- Die Anzahl der erzeugten Spikes *s*, die sich aus dem Füllstand der Spikeliste ergibt.
- Die Anzahl der von außen in das Netz eingespeisten Spikes *se*.
- Die Anzahl der in der Erregungsphase ausgeführten Erregungen *e*.
- Die Anzahl der Treffer im EIB-Cache, wobei hier die Anzahl der aus dem Cache bedienten Erregungen h_1 oder die Anzahl der präsynaptischen Spikes h_2, die zu einem Cache-Treffer führen, gemeint sein kann. Mit h_1 und h_2 ergeben sich unterschiedliche Trefferraten, da EIBs, die sich im Cache befinden, und solche, die sich nicht dort befinden, unterschiedliche mittlere Längen haben können.

Der gesamte Rechenaufwand *AU* eines Zeitschritts läßt sich im wesentlichen aus diesen Parametern und dem konstanten Aufwand für die jeweilige Aktion ermitteln. Solche konstanten oder statischen Aufwandparameter sind:

- *A* für das Abklingen der Teilpotentiale, die Berechnung des Membranpotentials und den Vergleich mit der dynamischen Schwelle für ein einzelnes Neuron.
- *W* für den entsprechenden Aufwand bei Einsatz der Wiedervorlageliste, wobei *W* aufgrund der zusätzlichen Wiedervorlagezeitberechnung größer als *A* ist.
- *S* für die Erzeugung eines Spikes im Fall der Überschwelligkeit.
- *E* für eine einzelne Erregung eines Neurons.
- *B* für das Suchen des EIBs eines Neurons.
- *C* für das Einspeisen eines externen Spikes.

Zusätzlich tritt für einen Zeitschritt bei der Simulation ein Kommunikationsaufwand *KA* auf, der sich auf folgende konstante Aufwandblöcke verteilt:

- *N* für den Zugriff auf die Neuronendaten eines Neurons.

- *AL* für einen Zugriff auf Abklingliste und Tag-Speicher.
- *WL* für den entsprechenden Aufwand bei der Wiedervorlageliste.
- *SL* für den Aufwand eines Zugriffs auf die Spikeliste.
- *BSS* für einen Blockstartspeicherzugriff.
- *EIB* für einen Topologiespeicherzugriff.

Auf Basis dieser Daten ergibt sich dann ein Ausdruck für den Rechenaufwand *AU* eines Zeitschrittes.

$$AU = a \cdot A + s \cdot S + s \cdot B + e \cdot E + se \cdot C \qquad \text{(Gl. 3.25)}$$

In diesem Ausdruck stehen die ersten beiden Summanden für den Aufwand der Abklingphase und die weiteren Summanden für die Erregungsphase. In Anbetracht der Tatsache, daß die Anzahl der aktiven Neuronen *a* wesentlich größer als die Anzahl der Spikes *s* bzw. *se* ist (siehe Kap. 2.4) und diese Spikes wiederum zu einer wesentlich größeren Anzahl von Erregungen *e* führen, wird *AU* hauptsächlich durch den ersten und den vierten Summanden bestimmt. Wird statt der Abklingliste die Wiedervorlageliste benutzt, so wird der erste Summand durch $wv \cdot W$ ersetzt. Auch wenn $a \geq wv$ ist, muß beachtet werden, daß dieser Vorteil durch $W \geq A$ aufgewogen werden kann.

Zur Einschätzung eines Simulationsverfahrens ist außerdem die Anzahl der Zugriffe auf Speicher oder Kommunikationsressourcen *KA* relevant, wobei die einzelnen Zugriffe mit unterschiedlichem Aufwand verbunden sein können.

$$\begin{aligned} KA = {} & a \cdot (2 \cdot N + (1 + b) \cdot AL) + (s + se) \cdot SL \\ & + (s + se) \cdot (BSS + SL) + e \cdot (EIB + 2 \cdot N + c \cdot AL) \end{aligned} \qquad \text{(Gl. 3.26)}$$

Dabei steht der erste Summand für Zugriffe während der Abklingphase, in der jeweils eine Neuronenadresse aus der Abklingliste gelesen werden muß, die Neuronendaten dieses Neurons gelesen, modifiziert und geschrieben werden und ggf. ein Anteil *b* der Neuronenadressen wieder in die Abklingliste zurückgeschrieben wird. Der zweite Summand beschreibt die Einträge in die Spikeliste, wobei die erzeugten Spikes *s* und die externen Spikes *se* berücksichtigt werden. In der Erregungsphase erfolgen entsprechend dem dritten Summanden für alle Spikes in der Spikeliste Zugriffe auf diese Liste und den Blockstartspeicher. Der vierte und letzte Summand beschreibt die Zugriffe für die Erregungen, für die eine Zeile aus dem EIB-Speicher gelesen werden muß, ein Lese-Modifizier-Schreib-Zugriff auf die Daten des erregten Neurons erfolgt und ggf. die Neuronenadresse des erregten Neurons in die Abklingliste eingetragen wird (Anteil *c* der erregten Neuronen). Wiederum verursachen der er-

ste und der vierte Summand den Hauptaufwand, da *a* bzw. *e* wesentlich größer als *s* bzw. *se* sind. Es läßt sich auch leicht erkennen, daß die Zugriffe auf die Neuronendaten *N* besonders kritisch sind, insbesondere wenn berücksichtigt wird, daß es sich dabei um einen ganzen Datenblock handelt. Für den Einsatz der Wiedervorlageliste gelten diese Aussagen mit $a \geq wv$ analog.

Wird der EIB-Cache eingesetzt und der Aufwand für den Zugriff auf diesen Cache aus dem gesamten Kommunikationsaufwand herausgenommen, so reduzieren sich der dritte und vierte Summand in (Gl. 3.26), indem aus den jeweils ersten Faktoren die Cachetreffer herausgerechnet werden, so daß sich der Zusammenhang

$$KA = a \cdot (2 \cdot N + (1 + b) \cdot AL) + (s + se) \cdot SL + (s - h_2 + se) \cdot (BSS + SL) + (e - h_1) \cdot EIB + e \cdot (2 \cdot N + c \cdot AL) \quad \text{(Gl. 3.27)}$$

ergibt. Dabei wird angenommen, daß Erregungen über den Cache ohne den Umweg über eine allgemeine Spikeliste, den BSS und den allgemeinen EIB-Speicher sofort zu Erregungen führen.

Bezüglich der Zusammensetzung der aktuellen Abklingliste (*t*) aus den Einträgen des vorherigen Zeitschritts (*t*-1) läßt sich der Ausdruck

$$a(t) = b \cdot a(t-1) + c \cdot e(t-1) \quad \text{(Gl. 3.28)}$$

bilden. Dabei gibt der erste Summand die Zahl der in der Abklingphase gelesenen Neuronenadressen an, die aufgrund des noch nicht erreichten Ruhezustands wieder in die Abklingliste eingetragen werden, und der zweite Summand die Anzahl der Neuronen, die erregt wurden, sich im Ruhezustand befanden und daher neu in die Abklingliste aufgenommen wurden.

Die verschiedenen veränderlichen Parameter wurden mit Hilfe des Softwaresimulators durch die Simulation des Weitzelnetzes (siehe Kap. 2.4.1) mit dem zweiten Stimulus ermittelt. Falls sich dazu unterschiedliche Werte ergeben haben, sind die Simulationsergebnisse für das Stoeckernetz (siehe Kap. 2.4.2) bzw. das Brausenetz (siehe Kap. 2.4.3) angegeben. Die statischen Parameter sind jeweils von der verwendeten Simulationsplattform abhängig. Für die Softwaresimulationen sind sie nicht mit vertretbarem Aufwand ermittelbar, da hierzu die verwendeten Sun-Workstations bezüglich der Hardware ausgemessen werden müßten und für das Betriebssystem ein reproduzierbares meßbares Verhalten vorausgesetzt werden müßte. Für die im weiteren Verlauf vorgestellten Hardware-Architekturen sind diese Parameter hingegen - soweit sinnvoll - ermittelt worden. Insbesondere bei Hardware-Konzepten läßt die Kombination der ermittelbaren statischen Parameter mit den simulierten dynamischen Parametern recht sinnvolle Prognosen über eine zukünftige Realisierung des Konzepts

zu.

Im Rahmen der Simulationen ist die Menge der erzeugten Spikes *s* über 2000 Zeitschritte für das Weitzelnetz ermittelt worden. Der Verlauf ist nachfolgend graphisch dargestellt. Mittelwerte und Standardabweichungen wurden für die Zeitschritte 200 bis 2000 ermittelt, also nach dem Einschwingvorgang. Dieses Vorgehen wurde auch für die anderen ermittelten Parameter gewählt.

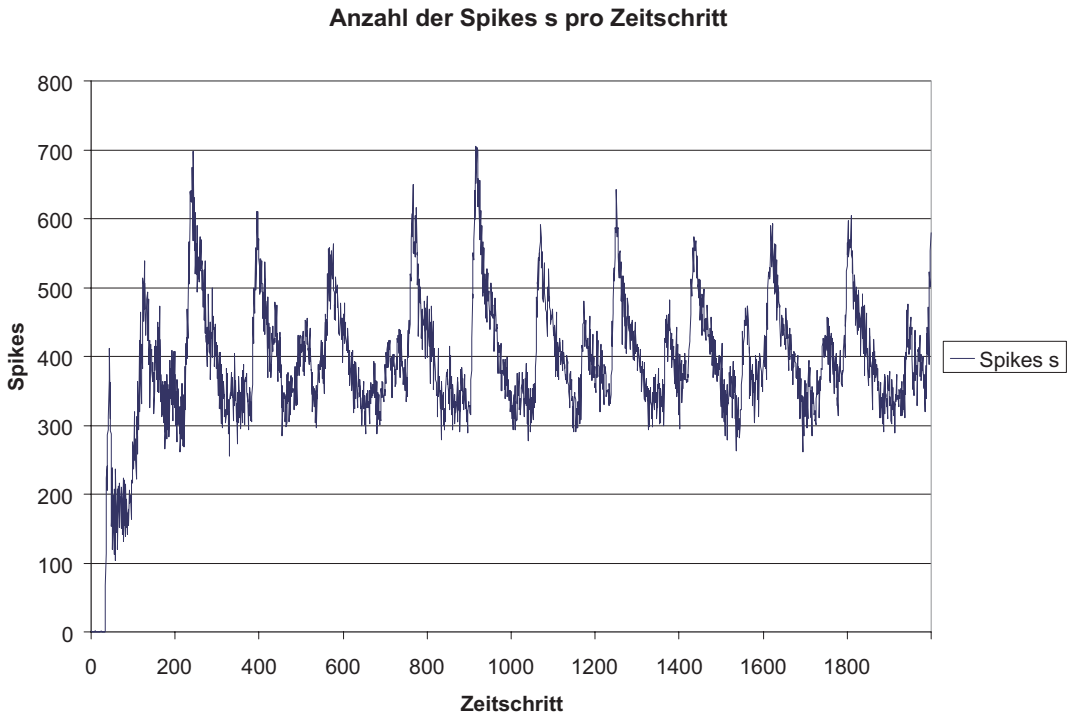

Abb. 3-7 Anzahl der emittierten Spikes s (Weitzelnetz)

In Abb. 3-7 sind die regelmäßigen Schwankungen der Spikemenge zu sehen, die durch das synchronisierte Feuern ganzer Neuronengruppen entstehen. Im Mittel feuern 404 Neuronen, die Standardabweichung beträgt 76. Die Menge der eingespeisten Spikes *se* beträgt im Mittel 49 Spikes mit einer Standardabweichung von 19, wobei das rhythmische Verhalten der erzeugten Spikemenge nicht auftritt.

Wird das Weitzelnetz mit dem Stimulus 1 gespeist (siehe Kap. 2.4.1), so ergeben sich 142 Spikes im Mittel mit einer Standardabweichung von 18. Das Brausenetz (siehe Kap. 2.4.3) erzeugt im Mittel 678 Spikes pro Zeitschritt mit einer Standardabweichung von 90. Beim Stoeckernetz (siehe Kap. 2.4.2) wurden 581 Spikes im Mittel mit einer Standardabweichung von 4 gemessen.

Weiterhin interessant ist, zu wievielen Erregungen *e* diese Spikes führen. Im Mittel werden beim Weitzelnetz 28.269 Erregungen ausgeführt, die Standardabweichung beträgt 3.078, ist

also im Verhältnis etwas geringer als bei den Spikes. Insgesamt schwanken diese Werte jedoch recht stark (siehe Abb. 3-8). Durch Verwendung der dynamischen senderorientierten Topologiespeicherung in Verbindung mit der Spikeliste müssen von den ca. 4 Mio. Verbindungen des Netzes also nur 0,7% tatsächlich berechnet werden. Aufgrund der geringeren Spikeanzahl *s* bei Verwendung des Stimulus 1 ergibt sich auch eine geringere Anzahl von Erregungen *e* pro Zeitschritt mit einem Mittelwert von 10.278 Erregungen und einer Standardabweichung von 1.437.

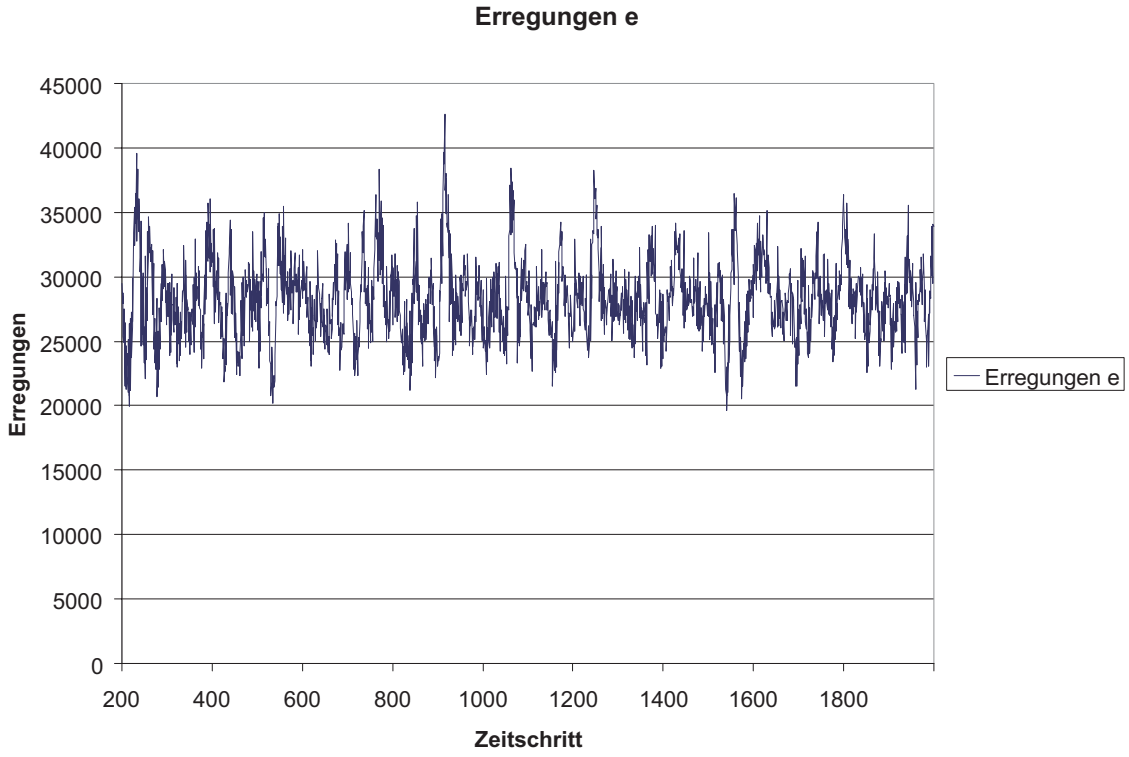

Abb. 3-8 Erregungen e in der Erregungsphase (Weitzelnetz)

Das Brausenetz wiederum zeigt einen Mittelwert von 322.257 Erregungen pro Zeitschritt mit einer Standardabweichung von 139.845. Diese auch unter Berücksichtigung der Netzgrößen recht hohe Anzahl von erregten Neuronen - 1,5% der Verbindungen werden pro Zeitschritt berechnet gegenüber 0,7% beim Weitzelnetz - resultiert aus der frühen Verarbeitungsstufe und der damit geringeren Reizspezifität des Brausenetzes. Die Gangliensignale werden an alle Lagen der simplen Neuronen weitergeleitet und führen dort zu Erregungen. Das Stoekkernetz verhält sich ähnlich zum Brausenetz. Es erfährt im Mittel 265.025 Erregungen bei einer allerdings sehr geringen Standardabweichung von 25.

Der Füllstand der Abklingliste *a* zeichnet sich in allen Simulationen durch eine wesentlich geringere Standardabweichung, d.h. durch einen glatteren Verlauf aus (Abb. 3-9). Beim Weitzelnetz betrug der mittlere Füllstand 16.993 Neuronenadressen mit einer Standardab-

weichung von 403. Das Ergebnis für den Stimulus 1 ist nahezu identisch. Bei einer Gesamtzahl von 114.401 Neuronen ergibt sich durch den Einsatz der Abklingliste gegenüber der Berechnung aller Neuronen in der Abklingphase eine Beschleunigung um den Faktor 6,73 bzw. eine Reduktion der Rechenzeit auf 14,9% der ansonsten notwendigen Rechenschritte. Beim Brausenetz fällt dieser Vorteil weitaus geringer aus. Von den 339.864 Neuronen müssen im Mittel 268.288 in der Aklingphase berechnet werden. Damit wird der Rechenaufwand nur auf 78,9% der Rechenschritte reduziert, die bei der Berechnung aller Neuronen notwendig wären. Die Standardabweichung liegt bei 1.422 und ist damit wie beim Weitzelnetz in Relation zu den Erregungen geringer, d.h. der Verlauf des Füllstands der Abklingliste ist glatter als der Verlauf der Erregungen pro Zeitschritt. Aufgrund der zur Ermittlung der Neuronenaktivität eingesetzten Leckintegratoren und ihrer Tiefpaßwirkung ist dieses Verhalten plausibel. Vor dem Hintergrund der wesentlich größeren Anzahl von Erregungen beim Brausenetz und der dort gewählten größeren Zeitkonstanten der Leckintegratoren ist auch der höhere Prozentsatz von aktiven Neuronen gegenüber dem Weitzelnetz nachvollziehbar.

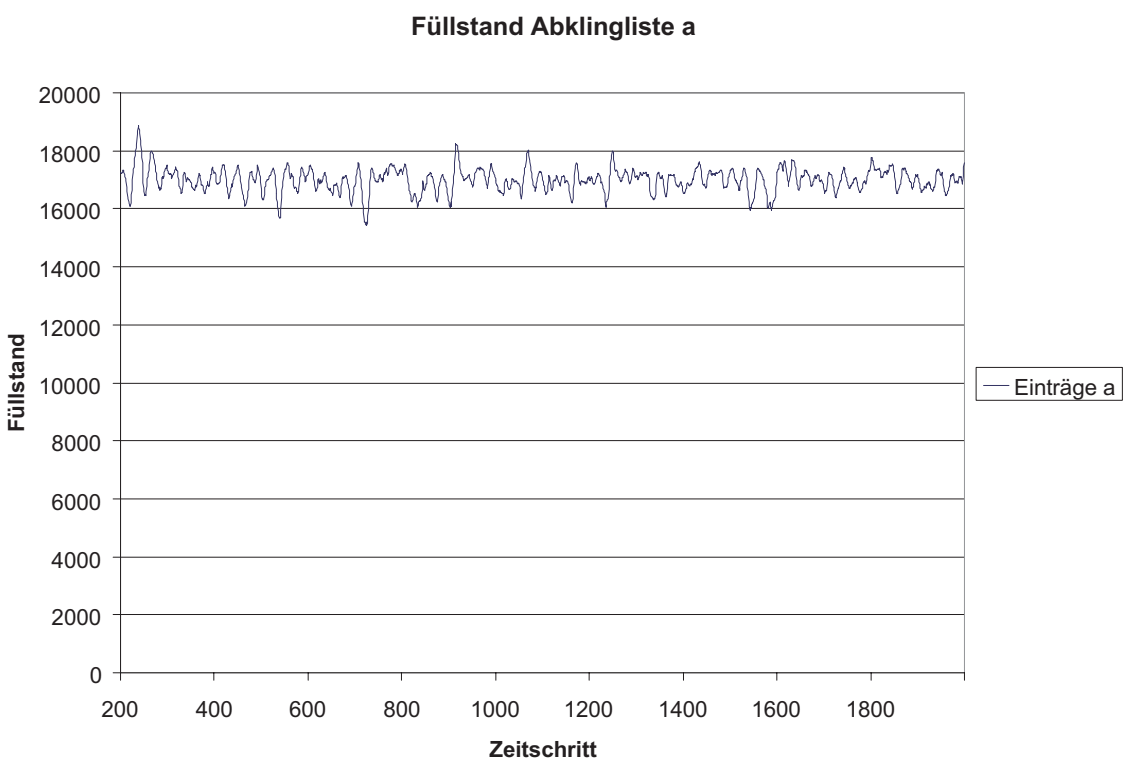

Abb. 3-9 Abzuklingende Neuronen a (Weitzelnetz)

Beim Stoeckernetz ergibt sich mit 13.621 aktiven Neuronen bei einer Standardabweichung von 26 eine Netzaktivität, die mit 50,2 % zwischen den Ergebnissen der beiden anderen Netze liegt.

Durch die Verwendung der Wiedervorlageliste (siehe Kap. 3.4.3) sollte die Anzahl der zu berechnenden Neuronen gegenüber dem Abklinglistenalgorithmus weiter gesenkt werden.

Für das Weitzelnetz ergibt sich hier jedoch ein recht ernüchterndes Resultat. Gegenüber 16.993 Neuronen, die bei der Abklingliste pro Zeitschritt berechnet werden müssen, fallen bei Verwendung der Wiedervorlageliste immer noch 15.393 Neuronen zur Berechnung an. Die Simulation der Abklingphase wird damit nur um den Faktor 1,1 beschleunigt. Dieser Faktor erhöht sich auch bei wesentlich längeren möglichen Ablagezeiträumen nicht signifikant. Der Grund für diese geringe Beschleunigung liegt darin, daß die Neuronen im Weitzelnetz ständig von anderen Neuronen erregt werden, so daß auch bei einer Ablage zur Wiedervorlage die Neuronen ständig vorzeitig aktualisiert werden müssen. Somit ergeben sich keine signifikanten Ablagezeiträume. Bei im Mittel 2.418 zur Wiedervorlage abgelegten Neuronen pro Zeitschritt erfahren gleichzeitig im Mittel 2.273 abgelegte Neuronen eine Erregung durch ein anderes Neuron, d.h. sie werden verfrüht wiedervorgelegt. In Abb. 3-10 wird dieser Sachverhalt über die Zeitschritte hinweg durch praktisch gleichlaufende Kurven verdeutlicht (submitted = neu abgelegte Neuronen, reset = erregte abgelegte Neuronen).

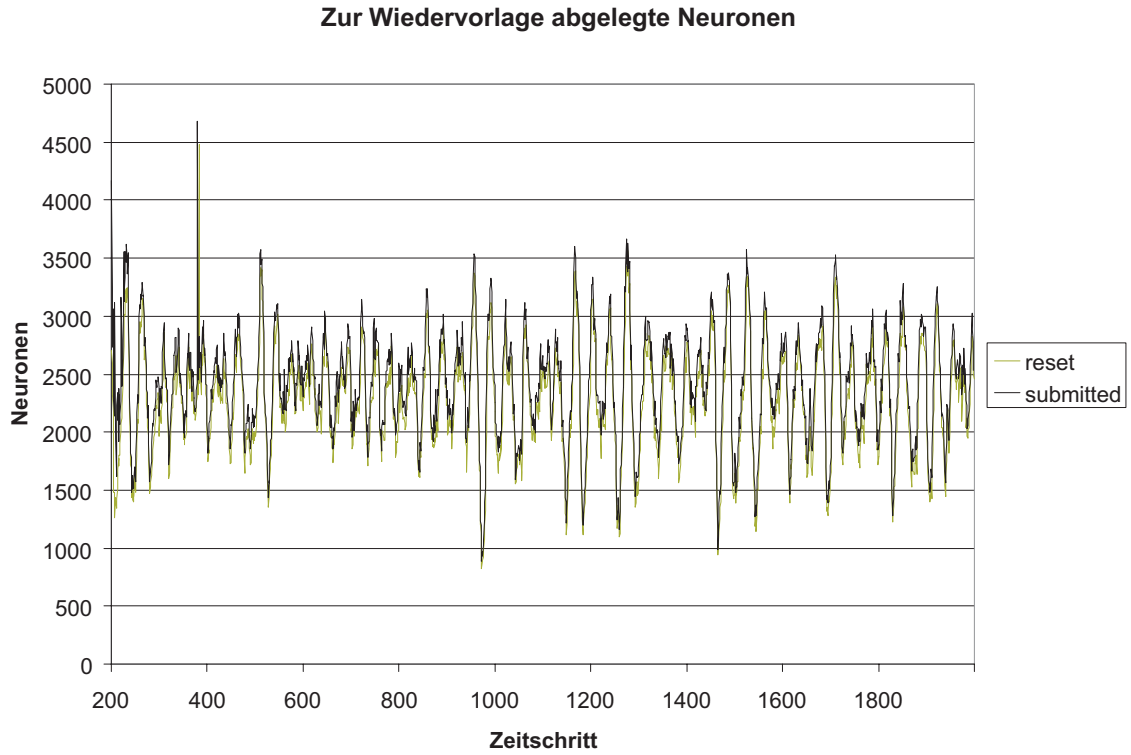

Abb. 3-10 Neu abgelegte (submitted) und verfrüht wiedervorgelegte (reset) Neuronen (Weitzelnetz)

Beim Brausenetz stellt sich dieser Sachverhalt etwas günstiger dar. Gegenüber 268.288 zu berechnenden Neuronen bei der Verwendung der Abklingliste sinkt diese Zahl bei der Wiedervorlageliste auf 160.858 bei einer Standardabweichung von 42.434. Es ergibt sich also im Mittel eine Beschleunigung um den Faktor 1,67 bei allerdings sehr starken Schwankungen. Gegenüber den insgesamt 339.864 Neuronen im Netz sind das immer noch 47%. Die

Schwankungen resultieren daraus, daß die filternde Wirkung der Leckintegratoren, deren Abklingzeit den Füllstand der Abklingliste bestimmt, durch die diskreten Wiedervorlageereignisse bei einer Erregung des abgelegten Neurons ersetzt werden. Da der Einsatz der Wiedervorlageliste neben einer reduzierten Anzahl zu berechnender Neuronen jedoch einen erhöhten Rechenaufwand für das einzelne Neuron nach sich zieht ($W \geq A$), erscheint der Einsatz beim Weitzelnetz aufgrund der geringen Reduktion der Anzahl der zu berechnenden Neuronen nicht sinnvoll. Das Stoeckernetz verhält sich in diesem Fall ähnlich dem Weitzelnetz.

Ein weiterer zu ermittelnder Wert ist die Trefferrate des EIB-Caches. Dazu wurde das in Kap. 3.4.3 vorgestellte Verfahren mit verschiedenen Cache-Größen simuliert. Zur Simulation kommen das Weitzelnetz und das Stoeckernetz in Frage, da das Brausenetz nur von externen Ganglien erregt wird und keine eigenen Verbindungen untereinander aufweist. Damit sind keine für das Caching geeigneten EIBs vorhanden.

Wird beim Weitzelnetz mit seinen insgesamt etwa 4 Mio. Verbindungen ein EIB-Cache mit 131.072 Einträgen benutzt, so läßt sich dort die gesamte während der Simulation benötigte Topologie nach und nach unterbringen. Der Cache wird also während der Simulation nicht geleert, so daß von im Mittel 28.269 Erregungen 26.472 (h_1) aus dem Cache bedient werden können. Die Trefferquote bezüglich der Erregungen beträgt damit mehr als 93%. Ist der Cache gefüllt, so werden in diesem Fall nur noch extern zugeführte Erregungen nicht aus dem Cache bedient. Wird der Cache kleiner gewählt (65.536 Einträge), so daß es zu regelmäßigen Leerungen kommt, wird mit 25.888 aus dem Cache bedienten Erregungen auch weiterhin eine hohe Trefferquote von 91% erreicht. Bezüglich der präsynaptischen Spikes, die über den Cache abgewickelt werden, ergibt sich ein Verhältnis von 404 Treffern (h_2) zu 453 präsynaptischen Spikes insgesamt bzw. bei kleinerem Cache von 388 zu 453, d.h. es ergeben sich Trefferquoten von 89% bzw. 85%. Mit dem Stoeckernetz werden die gleichen guten Trefferraten bei einem aufgrund der hohen Verbindungszahl größeren Cache erreicht. Grund für die hohen Trefferraten ist der konstante Eingangsreiz, der an die Netze angelegt wird. Dadurch bedingt bleibt die Aktivität im Netz auf lokale Bereiche beschränkt, so daß die für das Caching benötigte hohe Lokalität gewährleistet ist. Da ein bewegter Reiz nicht zur Verfügung stand, kann für einen solchen Fall keine Aussage getroffen werden. Vor dem Hintergrund, daß bei der Simulation mit dem kleineren Cache etwa alle 200 Zeitschritte der Cache geleert wurde, was wiederum 0,2 Sekunden im biologischen Vorbild entspricht, kann allerdings eine hohe Trefferquote auch für bewegte Reize erwartet werden.

Neben diesen qualitativen Aussagen über die verschiedenen Simulationsverfahren sind auch reine Zeitmessungen des sequentiellen Simulationsverfahrens von Interesse. Dazu wurden drei Simulatorversionen, in denen die Verfahren kombiniert wurden, auf einer Sun Ultra 60 Workstation gemessen. Für die erste Variante, die mit einer Abklingliste, der Spikeliste und der senderorientierten Topologiespeicherung arbeitet, wurde ein Mittelwert von 45 ms (Mil-

lisekunden) zur Berechnung eines Zeitschritts gemessen. Wird die Topologiespeicherung um einen EIB-Cache ergänzt, so ergibt sich eine Zeit von 40 ms. Dieser Wert ist recht erstaunlich vor dem Hintergrund eines aufwendigeren Algorithmus durch den zusätzlichen Cache, denn dieser Cache ist ja nicht schaltungstechnisch realisiert, sondern eine einfache Pufferverwaltung im sequentiellen Programm. Durch den Einsatz des EIB-Caches erhöht sich jedoch die Lokalität in diesem Programm, so daß wiederum der Cache der Sun eine höhere Trefferrate erreicht. Bei der Ultra 60 ist dieser Cache mit 4 MB so groß, daß die Simulation mit dem insgesamt 512kB großen EIB-Cache weitgehend im Cache der Sun abläuft, während dieses mit dem insgesamt 16 MB großen Topologiespeicher nicht möglich wäre. Auf einer Sun Sparcstation 5 mit nur 512kB Cache-Speicher tritt dieser Effekt dann auch nicht auf, die Simulation läuft im Verhältnis um 10% langsamer. Das durch die Erweiterung des Verfahrens um den EIB-Cache die Simulation nur um 10% langsamer wird, kann durch den sehr einfachen gewählten Algorithmus erreicht werden (siehe Kap. 3.4.3).

Wie sehr der Simulationsalgorithmus durch Erweiterungen verlangsamt wird, zeigt sich beim Ersatz der Abklingliste durch die Wiedervorlageliste, die einige zusätzliche Berechnungen benötigt. Obwohl etwa 10% weniger Neuronen berechnet werden müssen und die Erweiterung nur die Abklingphase betrifft, verlangsamt sich die Simulation des Weitzelnetzes auf 65 ms gegenüber den 45 ms mit der Abklingliste. Die Verlangsamung um den Faktor 1,44 durch den erhöhten Rechenaufwand für ein einzelnes Neuron ($W \geq A$) rechtfertigt also selbst mit der beim Brausenetz erreichten Reduktion der Anzahl der zu berechnenden Neuronen um den Faktor 1,67 den Einsatz der Wiedervorlageliste nicht. In der Tendenz zeichnet sich ab, daß Netze mit sehr hohen Zeitkonstanten für die Leckintegratoren (Brausenetz), also einer feineren Zeitschritteinteilung, eher von der Wiedervorlageliste profitieren, als solche mit kleinen Zeitkonstanten für die Leckintegratoren (Weitzel- und Stoeckernetz). Im weiteren wird die Wiedervorlageliste nicht mehr berücksichtigt.

Nachdem nunmehr ein sequentielles Simulationsverfahren vorliegt, das sich sowohl in Software als auch in Hardware abbilden läßt, stellt sich die Frage nach der Simulationsplattform. Aufgrund des rasanten Fortschritts in der kommerziellen Rechnertechnologie erscheint es zunächst sinnvoll, den vorhandenen optimierten Softwaresimulator auf einer Standard-Workstation oder einem PC einzusetzen. Wenn auf einer solchen Plattform in naher Zukunft eine ausreichende Simulationsgeschwindigkeit zu erreichen ist, entfällt die Notwendigkeit zu parallelen Ansätzen oder zum Einsatz von Spezialhardware. Um dieses zu überprüfen sind verschiedene Sun-Workstations zur Simulation eingesetzt worden. Den Simulationsergebnissen sind Benchmark-Werte dieser Maschinen gegenübergestellt worden, um abschätzen zu können, ob die Simulationsleistung in bezug auf PCNN proportional zur Prozessorleistung der Maschinen steigt. Dazu wurden die Werte der SPEC-Benchmark-Suite herangezogen, die bei der Standard Performance Evaluation Corporation [SPEC95] zu beziehen sind. Dieser Benchmark ist im Bereich der Ein-Prozessor-Standard-Rechner weit ver-

breitet und mißt insbesondere die Fließkomma- und die Festkommarechenleistung der einzelnen Prozessoren. Im folgenden Diagramm (Abb. 3-11) sind den gemessenen Simulationszeiten für die einzelnen Maschinen diejenigen Zeiten gegenübergestellt, die erreicht werden müßten, wenn das Verfahren vollständig mit der Prozessorleistung skaliert, d.h. die Werte der SPEC-Suite sind in fiktive Simulationszeiten umgerechnet. Hinter der Maschinenbezeichnung ist zusätzlich die Taktfrequenz des Prozessors angegeben.

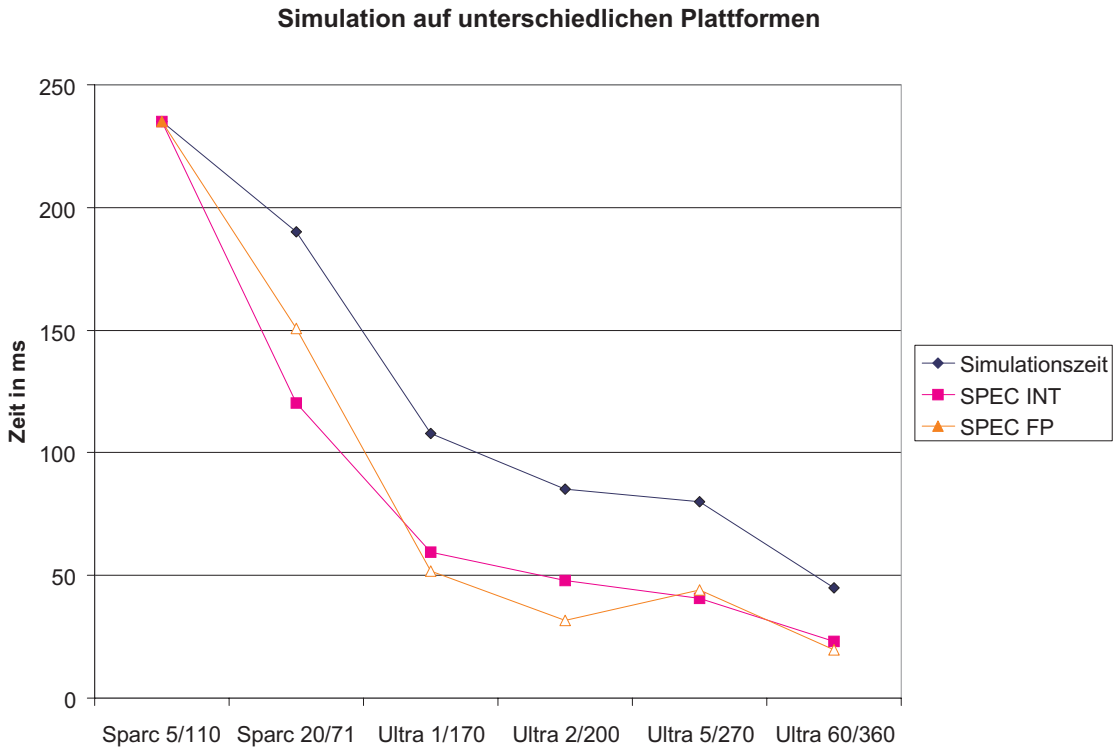

Abb. 3-11 Simulationszeiten im Vergleich zur Prozessorleistung

Tab. 3-2 Simulationszeiten auf unterschiedlichen Plattformen

Maschine	Zeit in ms	SPEC INT	SPEC FP
Sun Sparc 5 / 110	235	1,59	1,99
Sun Sparc 20 / 71	190	3,11	3,1
Sun Ultra 1 / 170	108	6,26	9,06
Sun Ultra 2 / 200	85	7,81	14,7
Sun Ultra 5 / 270	80	9,17	10,6
Sun Ultra 60 / 360	45	16,1	23,5
PC Pentium II / 266 (aus Berlin)	85	10,7	8,17

Tab. 3-2 Simulationszeiten auf unterschiedlichen Plattformen

Maschine	Zeit in ms	SPEC INT	SPEC FP
DEC Alpha 250/266 (aus Marburg)	140	4,18	5,78

Zum Vergleich und zur Einordnung der Sun-Maschinen sind in der Tabelle 3-2 zudem die SPEC-Werte eines gängigen PCs und einer DEC Alpha Workstation aufgenommen worden. Die Simulationsergebnisse für die DEC Alpha sind auf Basis des Weitzelnetzes, jedoch mit einem in der Eckhorn-Gruppe erstellten Simulationsprogramm ermittelt worden [SW97]. Die Werte für den Pentium-PC stammen aus der Arbeitsgruppe von Prof. Klar an der TU Berlin und sind anhand eines dem Weitzelnetz ähnlichen Netzes ermittelt worden [JRK96].

Die Simulationen zeigen, daß die Simulationsleistung einer Sun Ultra 60 für das Weitzelnetz mit 45 ms etwa eine Zehnerpotenz von der angestrebten Realzeitsimulation entfernt ist, für die im Fall der Bildverarbeitungs-PCNN 1 ms pro Zeitschritt gefordert wird [DDW98] [Fra97][SW97]. Dieser Abstand rechtfertigt nur noch bedingt den Einsatz einer Spezialhardware oder eines Parallelrechners, da eine Steigerung der Rechenleistung von Standardrechnern um den Faktor zehn in den nächsten Jahren zu erwarten ist. Es fällt allerdings auf, daß die Ultra 60 bei der Simulationszeit nicht voll von der gegenüber der Sparc 5 um eine Zehnerpotenz höheren Prozessorgeschwindigkeit profitieren kann, sondern nur um den Faktor fünf an Geschwindigkeit zulegt. Diese Tendenz deutet darauf hin, daß neben der Prozessorleistung auch andere Parameter die Simulationsleistung in bezug auf PCNN beeinflussen. Nach Untersuchungen in [JRS98] und aufgrund des Vergleichs von (Gl. 3.25) mit (Gl. 3.26) läßt sich ein erheblicher Teil dieser Abweichung mit der Speicherintensivität des Simulationsverfahrens begründen. Es werden überwiegend einfache Berechnungen auf einer großen Anzahl von Daten ausgeführt. Auch der Geschwindigkeitszuwachs der Ultra 60 gegenüber der Ultra 5 legt diese Begründung nahe, da aufgrund des 4 MB großen Cache-Speichers der Ultra 60 diese gegenüber der Ultra 5 mit 512 kB Cache einen schnelleren Speicherzugriff hat. Neben der Feststellung, daß die PCNN-Simulation nicht nur von der reinen Prozessorleistung abhängt, sondern auch von der Geschwindigkeit der Speicherschnittstelle, läßt sich aus Abb. 3-11 ersehen, daß der Einfluß der Integer-Rechenleistung (SPEC INT) größer ist, als derjenige der Floating-Point-Rechenleistung (SPEC FP). Diese Aussage erscheint jedoch selbstverständlich, da entsprechend Kap. 3.2 mit Festkommazahlen gerechnet wird.

Nachdem die Simulation des Weitzelnetzes auf konventionellen Ein-Prozessor-Maschinen vielversprechende Ergebnisse geliefert hat, erscheint die Entwicklung spezieller Lösungen zunächst überflüssig. Es ist jedoch zu beachten, daß Netze mit mehr als 1 Mio. Neuronen simuliert werden sollen, während das Weitzelnetz nur aus etwa 120.000 Neuronen besteht. Die Simulation des Brausenetzes mit etwa 340.000 Neuronen auf der Ultra 60 benötigt im Mittel 1.180 ms pro Zeitschritt. Dieser Sprung gegenüber dem Weitzelnetz mit 45 ms um mehr als eine Größenordnung rührt unter anderem daher, daß die 256 MB Arbeitsspeicher der Ma-

schine kaum für die Simulation ausreichen, d.h. es wird langsamer virtueller Speicher auf der Festplatte benutzt. Zudem sind weit mehr Neuronen aktiv, als dies beim Weitzelnetz der Fall ist. Hochgerechnet auf Netze mit 1 Mio. Neuronen ergeben sich Anforderungen, die durch konventionelle Standardrechner in der näheren Zukunft nicht zur erfüllen sind. Hinzu kommt die Tatsache, daß die Leistungsanforderungen gewöhnlich mit den gebotenen Möglichkeiten wachsen. Insbesondere für die Untersuchung von PCNN unter Realzeitbedingungen scheint also die Entwicklung von speziellen Lösungen weiterhin gerechtfertigt. Untersuchungen ohne Realzeitanforderung lassen sich hingegen durch konventionelle Rechner abdecken, da hier insbesondere der Speicherplatz relevant ist. Die benötigten 1-2 GB Hauptspeicher für Netze mit 1 Mio. Neuronen werden aktuell von Workstations geboten.

Bei der Bereitstellung einer speziellen Simulationsplattform für große PCNN bieten sich zwei Möglichkeiten an. Zum einen kann die sequentielle Simulation auf mehrere Rechenknoten verteilt werden. Diese Parallelisierung ist der Schwerpunkt der vorliegenden Arbeit und wird im weiteren noch ausführlich behandelt. Eine weitere Möglichkeit liegt in der schaltungstechnischen Realisierung des sequentiellen Verfahrens und in der Beschleunigung durch die Möglichkeiten der Rechnerarchitektur und Schaltungstechnik. Dieser Weg wurde mit dem SPIKE128k-System beschritten [FBH95][FH95][Fra97][HFS97][FHJ99].

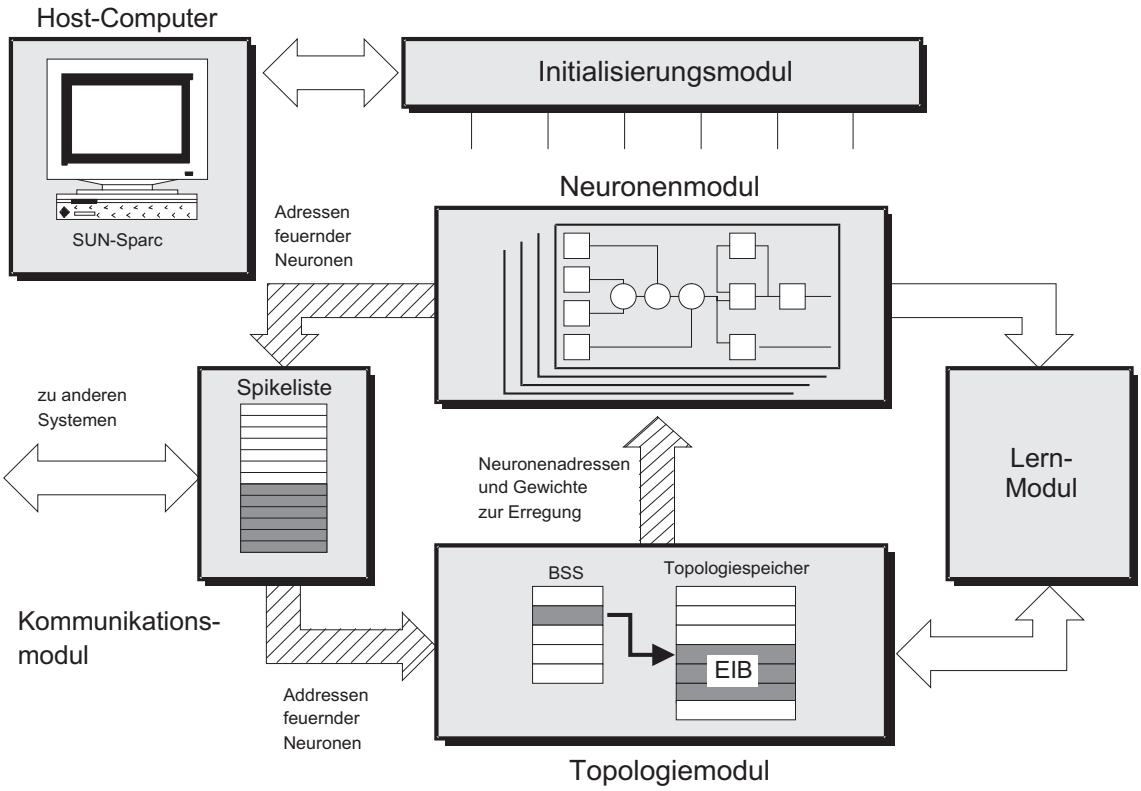

Abb. 3-12 Modularer Aufbau des SPIKE128k-Systems [Fra97]

Der SPIKE128k bildet den sequentiellen Algorithmus in Hardware ab, indem die Berech-

nungsschritte für ein Neuron bzw. eine Synapse in Form einer Verarbeitungspipeline aufgebaut werden. Die Schritte zur Behandlung der Neuronen in der Abklingphase sind in Form eines Neuronenmoduls aufgebaut, diejenigen zur Behandlung der Erregungsphase in Form eines Topologiemoduls. Der Datenaustausch zwischen den Modulen erfolgt auf Basis präsynaptischer Spikes (sogenannte *PreSpikes*) über eine Spikeliste auf einem Kommunikationsmodul bzw. durch die Übertragung postsynaptischer Spikes (sogenannte *PostSpikes*) vom Topologiemodul zum Neuronenmodul. Durch ein Initialisierungsmodul werden die Speicher des Systems angesprochen. Im Rahmen von [SH99][Sch00] wurde der SPIKE128k um ein Lernmodul bzw. ein integriertes Lern- und Topologiemodul ergänzt. Mechanisch sind die Module zu einem VME-Bus-System integriert.

Eine hohe Simulationsgeschwindigkeit wird beim SPIKE128k durch den Einsatz von Pipelining und Parallelität erreicht. Pipelining wird angewendet, wenn sich ein Rechenschritt in mehrere Unterschritte aufteilen läßt. Für jeden dieser Unterschritte wird dann eine eigene Berechnungseinheit bereitgestellt, die ihr Zwischenergebnis puffert und an den nachfolgenden Unterschritt weiterleitet, so daß eine Pipeline entsteht. Im Idealfall benötigt jedes der Pipelineelemente für seinen Unterschritt nur einen einzigen Takt des Gesamtsystems, d.h. das Element bekommt in jedem Takt eine neue Berechnung und gibt in jedem Takt ein Ergebnis an die Nachfolgeeinheit weiter. Bei einer Gesamtberechnung von 10 Schritten kann eine Pipeline mit einer Tiefe von 10 nach einer Latenzzeit von 10 Takten in jedem Folgetakt ein neues Ergebnis liefern bzw. eine neue Berechnung annehmen. Sequentielle Anteile beim PCNN-Algorithmus sind zum Beispiel die Schritte zur Berechnung des Membranpotentials und zum Vergleich mit der dynamischen Schwelle. Parallelität kann z.B. durch das gleichzeitige Abklingen aller Teilpotentiale eines Neurons erreicht werden, d.h. für jedes dieser Teilpotentiale ist dann eine eigene Arithmetikeinheit vorhanden. Die Pipeline des Neuronenmoduls des SPIKE128k kann alle Berechnungen der Abklingphase für ein Neuron in einem Takt durchführen. Zur Steuerung der Abklingphase wird die Abklingliste benutzt, so daß in jedem Takt des Systems ein Neuron aus der Abklingliste verarbeitet wird. Die Pipeline des Topologiemoduls führt pro Takt eine Erregung durch.

Schaltungstechnisch ist der SPIKE128k als Prototypsystem auf Basis programmierbarer Logik, kommerzieller Arithmetikeinheiten und DRAM bzw. SRAM-Speichern aufgebaut. Der Pipelinetakt des Systems beträgt 10 MHz. Auf Basis dieser Daten und der Simulationsergebnisse des Weitzelnetzes läßt sich mit (Gl. 3.25) folgende theoretische Simulationsgeschwindigkeit für einen Zeitschritt ermitteln:

$$AU = 16993 \cdot 100ns + 28269 \cdot 100ns = 4,52ms \qquad \text{(Gl. 3.29)}$$

Zu beachten ist, daß die Behandlung der Spikeliste durch das Kommunikationsmodul parallel zu den Berechnungen erfolgt und somit als Vor- oder Nachfolgestufe der jeweiligen Be-

rechnungspipeline angesehen werden kann. Gelingt also innerhalb der 4,52 ms eines Zeitschritts die Einspeisung der externen Spikes *se* und die Ausgabe der Ergebnisspikes, so wird die theoretische Geschwindigkeit erreicht. Tatsächlich gelingt die Abwicklung der Kommunikation allerdings nicht genügend schnell, so daß die gemessene Geschwindigkeit bei 11 ms pro Zeitschritt liegt. Der Grund für diesen Engpaß liegt in der Art der Kommunikation und der gewählten Form des Kommunikationssystems. Bei der PCNN-Simulation fallen sehr geringe Mengen von präsynaptischen PreSpikes an. Beim Weitzelnetz sind dies etwa 450 Spikes pro Zeitschritt. Jeder der Spikes wird als 3-Byte-Wort codiert, so daß etwa 1,5 kByte pro Zeitschritt übertragen werden müssen. Mit einer Zeitschrittdauer von 4,52 ms wären 330 kByte/s zu erreichen, was für moderne Kommunikationsschnittstellen kein Problem darstellt. Bei der PCNN-Simulation sind diese Datenraten jedoch in kleine Pakete für jeden Zeitschritt unterteilt, so daß Latenzzeiten bei der Verarbeitung relevant werden.

Der SPIKE128k ist an das Sun-Hostsystem über einen Transputerlink mit einer theoretischen Datenrate von 20 Mbit/s angebunden. Transputer sind spezielle Mikroprozessoren mit einer integrierten Linkschnittstelle zur Kommunikation. Auf dem Kommunikationsmodul befindet sich ein solcher Transputer und in der Sun ist eine weitere Karte mit einem Transputer eingebaut. Software auf diesen Transputern verpackt die Spikes gemäß eines Protokolls und packt sie auf der Gegenseite wieder aus. Auf der Sun läuft eine weitere Software, die mit dem Transputer in der Sun kommuniziert. Durch die aufwendigen in Software realisierten Protokolle und die geringe Rechenleistung der Transputer entsteht damit eine Latenzzeit für jedes Datenpaket. Daher verlangsamt die hohe Anzahl kleiner Datenpakete die gesamte Simulation um den Faktor 2,4. Damit ist der SPIKE128k jedoch auch heute noch signifikant schneller als verfügbare Standard-Rechner. Da das System auf Basis der Transputerlinks einfach mit weiteren Systemen gekoppelt werden kann, bietet es prinzipiell auch für größere Netze eine ansonsten nicht verfügbare Simulationsleistung.

Die Abbildung des Simulationsverfahrens auf eine Pipeline, wie sie im SPIKE128k realisiert ist, stellt annähernd das erreichbare Optimum aus Sicht der Rechnerarchitektur dar. Eine weitere Verbesserung wäre nur noch durch die Kombination von Abkling- und Erregungsphase in einer einzigen Pipeline bzw. durch die parallele Bearbeitung dieser beiden Phasen zu erreichen. Dieser Weg wurde im MASPINN-Konzept [SMJ98] in Berlin beschritten, das im weiteren Verlauf dieser Arbeit noch vorgestellt wird. Nachteil einer solchen Verschränkung der Simulationsphasen ist, daß keine Zeit für das Lernen bleibt, das beim SPIKE128k parallel zur Abklingphase erfolgt, da dann der Topologiespeicher während der gesamten Simulationszeit von den Berechnungen der Erregungsphase blockiert ist. Neben weiteren algorithmischen Verbesserungen, wie dem Einsatz der Wiedervorlageliste, bleibt daher für eine Beschleunigung der sequentiellen Simulation nur noch die Erhöhung des Pipelinetaktes bis zur Erreichung des schaltungstechnisch möglichen Maximums.

Mit einem solchen Vorgehen steigt jedoch auch der Entwicklungs- und Inbetriebnahmeauf-

wand, der schon beim SPIKE128k-System erheblichen Umfang erreicht hat. Aus diesem Grund entstand im Nachhall der SPIKE128k-Entwicklung ein Interesse an einer mit weniger Aufwand zu implementierenden Lösung, die zudem flexibler in bezug auf Änderungen des Simulationsverfahrens sein sollte als die sehr starre dedizierte Hardware des SPIKE128k. Da einzelne kommerzielle Mikroprozessoren wie gezeigt nicht die benötigte Simulationsleistung bieten, entstand ein Interesse an der Parallelisierung der Simulation. Dabei waren die folgenden Randbedingungen einzuhalten.

- Das parallele System soll in der Lage sein, 1 Mio. Neuronen mit einer Simulationszeit in der Größenordnung von 1 ms pro Zeitschritt zu simulieren. Die Perspektive zu noch höheren Neuronenanzahlen soll gegeben sein.
- Als Rechenknoten sollen kommerziell erhältliche Prozessoren verwendet werden, um den Entwicklungsaufwand zu reduzieren und vom mikroelektronischen Fortschritt profitieren zu können. Mittels solcher Prozessoren läßt sich zudem ein wesentlicher Anteil des Simulators in Software abbilden, was die Flexibilität erhöht und den Entwicklungsaufwand weiter reduziert.
- Alle weiteren Komponenten des Systems sollen weitgehend auf Standardkomponenten basieren und mit einem Minimum an programmierbarer Logik auskommen. Der Einsatz programmierbarer Logik ermöglicht gegenüber festverdrahteten Realisierungen ebenfalls mehr Flexibilität bei Änderungen und vereinfacht zudem den prototypischen Aufbau.
- Der Systemaufbau soll kompakt sein, d.h. für 1 Mio. Neuronen nicht mehr VME-Module als der SPIKE128k (4 Module) benötigen. Durch eine hohe Integration soll der Schnittstellenaufwand weitgehend von den Platinen in Bausteine verlagert werden und somit die Anzahl der Bauteile gering gehalten werden.

Aus diesen Vorgaben und den in diesem Kapitel dargestellten Ergebnissen der sequentiellen Simulation ergeben sich direkt Anforderungen an den Rechenknoten. Der zum Einsatz kommende Rechenknoten muß

- kompakt aufgebaut sein, um eine hohe Integrationsdichte auf den Systemplatinen zu erreichen,
- möglichst über einen eigenen Speicher verfügen, da die Simulation sehr speicherintensiv ist,
- über Kommunikationsressourcen verfügen, die den Aufbau eines Parallelrechnersystems unterstützen, und
- die benötigte Rechenleistung bieten.

Auf Basis dieser Anforderungen wurden die weiteren Untersuchungen im Rahmen dieser

Arbeit durchgeführt. Dabei war das Ziel die Ermittlung der nötigen Grundlagen der parallelen Simulation großer PCNN und darauf aufbauend der Entwurf eines Konzepts für den geforderten parallelen Spezialrechner, das im weiteren vorgestellt und durch Simulationen bezüglich der möglichen Leistung einer Realisierung abgeschätzt wird.

Kapitel 4 - Methoden der parallelen Simulation

Die parallele Bearbeitung großer Problemstellungen auf mehreren Berechnungsknoten ist ein häufig genutzter Weg, die immer wieder erreichten Grenzen der Leistungsfähigkeit auch aktueller einzelner Prozessoren zu überwinden. Die auftretenden Probleme bei der Parallelisierung der Problemberechnung bieten der Informatik ein weites Feld für Forschungen. Um solche parallelisierten Berechnungen durchführen zu können, sind verschiedene Typen von speziellen Parallelrechnern entwickelt worden.

Auch zur Simulation neuronaler Netze sind aufgrund der Komplexität der Simulationsaufgabe verschiedene Ansätze zur Abbildung auf parallele Recheneinheiten entwickelt worden [NS92][SGS96][Mis97][Res99]. Vielversprechend erscheint eine solche Parallelisierung, da die Informationsverarbeitung im biologischen Vorbild massiv parallel erfolgt. In Gehirnen wird offenbar die Information von einer sehr großen Anzahl weitgehend gleichartiger und einfacher Nervenzellen verarbeitet [HW62]. Die Simulation des neuronalen Modells auf einzelnen Prozessoren wird realisiert, indem die an sich parallele Verarbeitung durch ein sequentielles Abarbeiten der einzelnen Einheiten ersetzt wird. Eine Parallelisierung dieser Simulation kann durch paralleles Abarbeiten jeder einzelnen Einheit oder von Gruppen solcher Einheiten erfolgen. Da zum einen die Rechenknoten paralleler Computer eine weitaus höhere Rechenleistung aufweisen, als üblicherweise für eine einzelne Einheit des neuronalen Netzes benötigt wird, und zum anderen in solchen Parallelrechnern weit weniger Rechenknoten vorhanden sind als Einheiten in einem neuronalen Netz, erfolgt die Parallelisierung gewöhnlich durch das Verteilen ganzer Gruppen von Elementen des neuronalen Netzes und durch das sequentielle Berechnen dieser Gruppen auf den Rechenknoten, so daß sich eine Mischung aus parallelem und sequentiellem Simulationsanteil ergibt. Auf die Formen der Parallelität bei der Simulation neuronaler Netze wird im weiteren noch eingegangen.

Die Tatsache, daß ganze Teile des neuronalen Netzes auf einem Rechenknoten berechnet werden, führt allerdings auch zum Hauptproblem der parallelen Simulation neuronaler Netze. Die Informationsverarbeitung in einem solchen Netz erfolgt nämlich nicht nur durch die einfachen Berechnungen der einzelnen Neuronen sondern insbesondere auch durch ausgiebige Kommunikation. Gehirne bestehen aus weitaus mehr Verbindungen zwischen Neuro-

nen als aus Neuronen selbst [Hub90]. Auf einem Parallelrechner ist jedoch eine große Anzahl von Neuronen über wenige physikalische Kommunikationskanäle mit einer großen Anzahl Neuronen auf einem anderen Rechenknoten verbunden. Parallelrechner sind offenbar gut geeignet, wenn ein Problem mit einem hohen Rechenanteil und einem geringen Kommunikationsanteil parallelisiert wird, während neuronale Netze auf einem hohen Kommunikationsanteil und einem geringen Rechenanteil basieren. Damit sind die inhärent parallelen neuronalen Netze nicht sonderlich gut zur Berechnung auf konventionellen Parallelrechnern geeignet. Es werden also Methoden benötigt, die vor diesem Hintergrund die parallele PCNN-Simulation sinnvoll machen.

Bevor auf die parallele Simulation im allgemeinen und auf die speziellen Ausprägungen in bezug auf die PCNN eingegangen wird, soll zunächst das parallele Grundverfahren erläutert werden, auf dem diese Arbeit basiert (siehe Abb. 4-1). Im weiteren wird dieses Verfahren in den Rahmen der parallelen Simulation eingeordnet und seine Wahl begründet. Daran anschließend werden die Strategien erläutert, mit denen die auftretenden Probleme gelöst werden, so daß abschließend ein Satz von geeigneten Simulationsmethoden für die parallele PCNN-Simulation zur Verfügung steht.

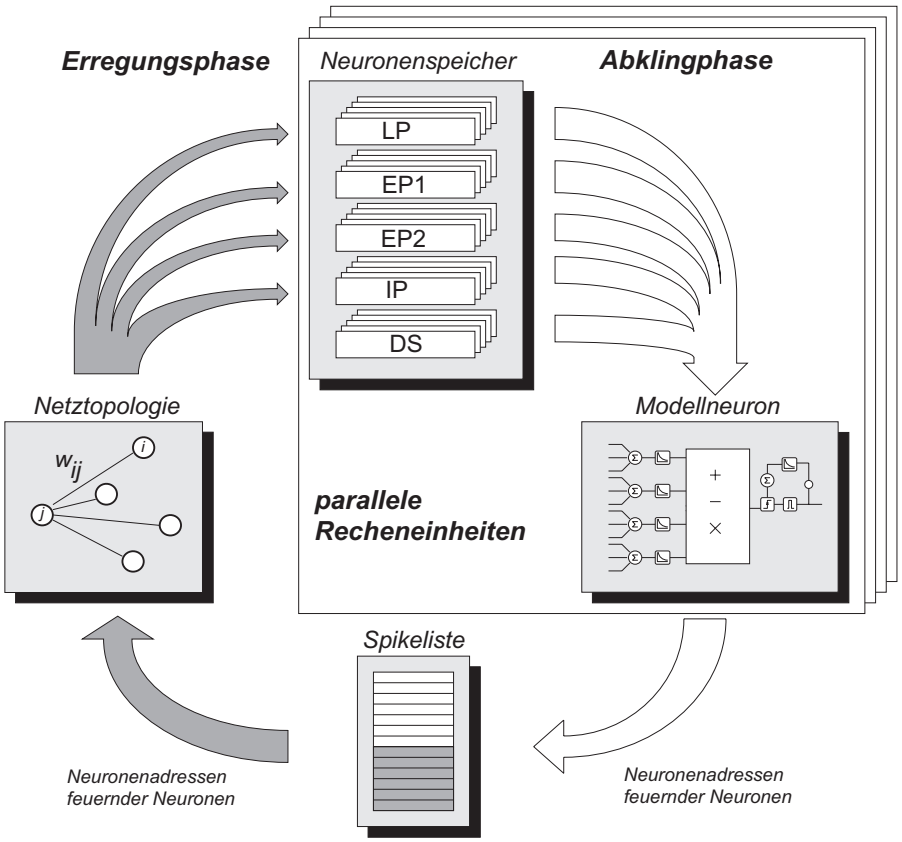

Abb. 4-1 Paralleles Grundverfahren zur PCNN-Simulation durch Verteilen der Neuronen auf parallele Recheneinheiten

Die Parallelisierung wird durch die Verteilung der Neuronen auf parallele Recheneinheiten realisiert. Diese Verteilung erfolgt durch die Aufteilung des Neuronenspeichers. Auf den Neuronendaten, die einem Rechenknoten zugewiesen wurden, können dann unabhängig von den anderen Rechenknoten die Berechnungen der Abklingphase durchgeführt werden. Dieser Teil der Parallelisierung erscheint einfach und naheliegend. Variationen der Verfahren werden sich im weiteren durch die Behandlung der Netztopologie, die global für alle Rechenknoten oder lokal auf den Knoten abgelegt sein kann, sowie durch die lokale oder globale Realisierung der Spikeliste ergeben. Je nach gewählter Lösung resultieren daraus unterschiedliche Verfahren der Kommunikation.

Die Synchronisation der Simulation erfolgt durch das zeitgleiche Eintreten der Rechenknoten in die Abkling- und Erregungsphase und damit durch den zeitgleichen Zeitschrittwechsel. Möglichkeiten, diese starre Synchronisation aufzuweichen, werden im weiteren dargestellt. Durch die starre Synchronisation bedingt müssen langsame Rechenknoten auf schnellere warten, so daß die Frage der Lastverteilung für den Erfolg der Parallelisierung außerordentlich wichtig wird. Dazu sind insbesondere Verfahren zur geeigneten Verteilung der Neuronen auf die Recheneinheiten erforderlich. Diese Verteilung bestimmt zudem den Kommunikationsaufwand zwischen den Rechenknoten entscheidend mit. Zur Lösung dieser Aufgabe werden Methoden der Netzpartitionierung vorgestellt.

4.1 Taxonomie paralleler Simulationssysteme

Nachdem nun das gewählte Simulationsverfahren bekannt ist, soll es aus verschiedenen Blickwinkeln beleuchtet werden und seine Wahl soll im Vergleich zu anderen Ansätzen näher begründet werden. Insbesondere erfolgt in diesem Zusammenhang die Einführung wesentlicher Begriffe aus dem Bereich der Parallelverarbeitung. Auf diesen Begriffsdefinitionen und dieser Taxonomie basierend werden dann in weiteren Kapiteln nähere Ausführungen zu Details des gewählten Verfahrens gegeben.

Bei der Parallelisierung zeitdiskreter Simulationen ergeben sich grundsätzlich zwei Ansätze. Zum einen kann die gesamte Abfolge der Zeitschritte in mehrere Blöcke zerlegt werden, die dann parallel berechnet werden. Dieser Ansatz wird als *zeitparallel* (*time parallel)* bezeichnet [Fer95]. Er ist jedoch nur für solche Simulationen geeignet, in denen das Ergebnis späterer Zeitschritte nicht von früheren Zeitschritten abhängt und zudem kein chronologisch korrekter Strom von Ergebnissen benötigt wird. Bei der Simulation von PCNN in realen Szenarien ist gerade das Gegenteil der Fall, so daß dieser Ansatz ausscheidet. Die zweite Möglichkeit besteht in der Ausnutzung von Parallelität in den Berechnungen eines Zeitschritts, d.h. innerhalb des Modells (*raumparallel - space parallel*). Durch die Verteilung der Neuronen im gewählten Simulationsverfahren wird dieser Ansatz verfolgt.

Grundsätzlich ergeben sich auch bei diesem Ansatz verschiedene Formen der Parallelität, zwischen denen Überschneidungen auftreten [NS92][Zel94].

- Im Fall der *Musterparallelität* wird auf mehreren Recheneinheiten das gleiche neuronale Netz berechnet, wobei jedem der Netze ein anderer Eingangsreiz zugeführt wird. Diese Art der Parallelisierung ist besonders einfach zu realisieren, insbesondere entsteht keine Kommunikation zwischen den Rechenknoten. Für das im Rahmen dieser Arbeit untersuchte Problem der echtzeitnahen PCNN-Simulation ist diese Variante uninteressant, da ein Rechenknoten wie gezeigt nicht über die nötige Rechenleistung zur Simulation eines vollständigen Netzes verfügt und daher gerade die Simulation eines Netzes parallelisiert werden soll.

- Die *Phasenparallelität* läßt sich im Grenzbereich zwischen einer zeitparallelen und einer raumparallelen Simulation ansiedeln. Im SPIKE128k-System wird teilweise phasenparallel simuliert, indem die Berechnungen für ein einzelnes Neuron in Form einer Pipeline parallel durchgeführt werden. Das im weiteren noch geschilderte MASPINN-System [SMJ98] geht sogar soweit, die Berechnung der Abkling- und Erregungsphase gleichzeitig durchzuführen. Diese Systeme verwenden hierzu jedoch eine dedizierte Hardware-Pipeline. Werden kommerzielle Prozessoren - ob in einem Spezialrechner oder einem Standard-Parallelrechner - zu einer solchen Pipeline kombiniert, so müssen große Datenmengen zwischen den Rechenknoten bewegt werden. Insbesondere der Zugriff auf die Neuronendaten N aus (Gl. 3.26) stellt hierbei einen kritischen Engpaß dar, da diese Daten durch alle Recheneinheiten der Pipeline geschleust werden müßten. Ein solcher Austausch der Aktivitätsdaten zwischen den Recheneinheiten steht zudem im Widerspruch zur pulscodierten Kommunikation im biologischen Vorbild. Im Rahmen der Betrachtungen über die Kommunikation im parallelen Simulator wird auf die Problematik dieser Möglichkeit noch eingegangen.

- Das gewählte Verfahren zur parallelen PCNN-Simulation verfolgt im wesentlichen den Ansatz der *Neuronenparallelität*. Da auch die Berechnungen zur Modifikation der Neuronenparameter aufgrund einer Erregung parallel auf den die jeweiligen Neuronen beheimatenden Prozessoren erfolgen, nutzt dieser Ansatz zumindestens z.T. auch *Synapsenparallelität*. Wenn nicht jedes Neuron parallel berechnet wird, so nennt sich der Ansatz mitunter auch *Ebenen-* oder *Lagenparallelität*. Dieser Ansatz erscheint aufgrund biologischer Plausibilität sinnvoll, da auch das Gehirn Neuronen und Synapsen parallel vorhält. Auch die Betrachtung des Kommunikationsaufwandes aus (Gl. 3.26) legt diesen Ansatz nahe, da der Neuronenspeicher, der besonders viele Zugriffe erfährt, auf die parallelen Einheiten verteilt wird und der Zugriff somit parallel erfolgen kann. Zur Parallelisierung des zweiten großen Blocks von Zugriffen auf Daten werden im Abschnitt über die Methoden der Netztopologiedarstellung weitere Ausführungen gemacht.

Neben den grundsätzlichen Formen der Parallelisierung werden verschiedene Stufen der Parallelität unterschieden. Im Zusammenhang mit dem gewählten Simulationsverfahren zur parallelen PCNN-Simulation sind dieses insbesondere verschiedene Stufen der parallelen ereignisbasierten Simulation [Fer95], die im folgenden entsprechend ihrer Feinkörnigkeit aufgelistet sind. Die Liste beginnt mit grobkörnigen (*large* oder *coarse grain*) und endet mit feinkörnigen (*fine grain*) Verfahren.

- Zunächst kann eine Parallelisierung auf der *Applikationsstufe* erfolgen, d.h. es wird auf verschiedenen Daten jeweils der gleiche Algorithmus berechnet. Dieses Verfahren entspricht bei der Neuronale-Netze-Simulation der Musterparallelität, die in der obigen Aufstellung erläutert und für die PCNN-Simulation als irrelevant eingeordnet wurde.
- Die *Subroutinenstufe* entspricht der Phasenparallelität in obiger Aufstellung. Da der Simulationsalgorithmus im allgemeinen weniger einzelne Stufen aufweist als das zu simulierende Netz Elemente enthält, können mit dieser Art der Parallelität weniger Aufgaben generiert werden, als mit dem gewählten Ansatz und somit auch weniger Rechenknoten beschäftigt werden. Allerdings läßt sich der Ansatz mit einer neuronenparallelen Simulation verknüpfen.
- Die Simulation auf *Komponentenstufe* kann durch neuronenparallele oder synapsenparallele Simulation erfolgen, da Neuronen und Synapsen die Elemente des Netzes bilden. Dieser Stufe ist der gewählte Ansatz zunächst zuzuordnen. Da jedoch nicht alle Neuronen und Synapsen in jedem Zeitschritt berechnet werden, sondern die ereignisgesteuerten Verfahren, die sich in den Untersuchungen aus Kap. 3.5 als erfolgreich erwiesen haben, zur Anwendung kommen sollen, ergeben sich Zuordnungen zu zwei weiteren Stufen, die sich mit der Behandlung der Ereignislisten beschäftigen.
- Dabei kann die parallele Simulation auf der *Ereignisstufe* auf Basis einer *zentralen Ereignisliste* erfolgen. Eine solche zentrale Ereignisliste ist die globale Spikeliste des gewählten Verfahrens.
- Als weitere Verfeinerung kann jedoch auch mit Hilfe einer *dezentralen Ereignisliste* simuliert werden, auf die ein paralleler Zugriff erfolgt. Eine solche dezentrale Ereignisliste ist die Abklingliste bzw. Wiedervorlageliste im gewählten Verfahren. Methoden zur Realisierung des Verfahrens mit einer dezentralen Spikeliste werden im weiteren noch vorgestellt.

Im Rahmen der nebenläufigen Verfahren werden neben der Parallelisierung des Simulationsproblems in seinen Abstufungen auch verschiedene Arten der Implementierung unterschieden. Zunächst einmal kann die Umsetzung *parallel* oder *verteilt* erfolgen [Fer95]. Dazu kann zwischen einer parallelen *synchronen* Bearbeitung des Problems durch mehrere Recheneinheiten oder einer verteilten *asynchronen* Verarbeitung unterschieden werden. Die

Abgrenzungen sind wie üblich unscharf und stark durch die Hardware der Simulationsplattform beeinflußt. Parallele Rechnersysteme werden entsprechend [Fly66] folgendermaßen klassifiziert:

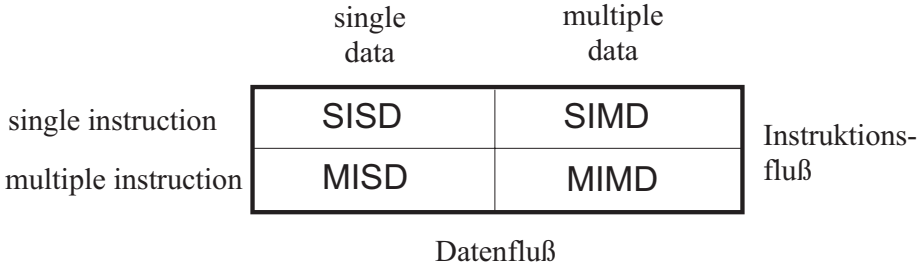

Abb. 4-2 Taxonomie paralleler Rechensysteme [Fly66]

Dabei wird unterschieden, ob das System verschiedene Daten zugleich bearbeitet (single data versus multiple data) und ob verschiedene Befehle gleichzeitig ausgeführt werden (single instruction versus multiple instruction). Parallele Verfahren wie das gleichzeitige Abklingen aller Teilpotentiale eines Neurons im SPIKE128k sind typische Beispiele für den SIMD-Ansatz und entsprechend der Klassifizierung [Fer95] parallele Rechenmodelle. Das vorgeschlagene Verfahren zur parallelen PCNN-Simulation soll hingegen auf mehreren kommerziellen Rechenknoten simuliert werden, die zwar zeitschrittsynchron aber nicht rechenschrittsynchron arbeiten. Jeder Rechenknoten berechnet zudem andere Neuronen, so daß es sich um eine typische MIMD-Architektur handelt, die gemäß [Fer95] als verteilte Simulation eingeordnet werden kann.

Eine weitere Klassifizierung von Berechnungsmodellen [Fra97] befaßt sich mit den Mechanismen, die auf höherer Abstraktionsebene die Algorithmen beschreiben:

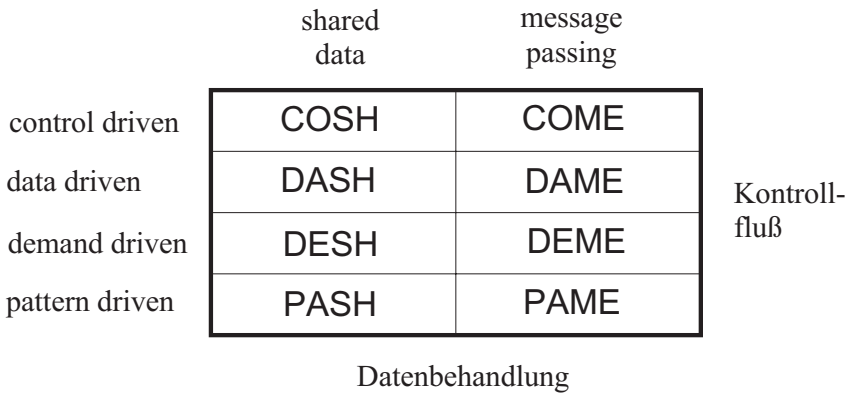

Abb. 4-3 Realisierung paralleler Berechnungsmodelle [Fra97]

Dabei werden Methoden des *Kontrollflusses* unterschieden, die Berechnungen steuern, sowie Methoden der *Datenbehandlung* bzw. des Datenaustausches zwischen parallelen Einhei-

ten. Auch hier sind die Grenzen unscharf, wobei sich je nach Abstraktionsstufe der Betrachtung des Verfahrens unterschiedliche Einordnungen ergeben. Ein Gehirn kann beispielsweise auf der Stufe der Neuronen als eine datengetriebene Architektur auf Basis von Botschaftenaustausch betrachtet werden (DAME). Werden funktionale Zellverbände betrachtet ergibt sich ein weiteres, komplexeres Berechnungsparadigma, nämlich die spezifische Reaktion auf ganze Muster, so daß eine Einordnung als PAME-System sinnvoll erscheint. Der SPIKE128k beruht auf der Abbildung eines sequentiellen Simulationsalgorithmus in Hardware (siehe Kap. 3.3), so daß er zunächst als COSH-Architektur eingeordnet werden kann. Der Simulationsalgorithmus arbeitet nun aber ereignisbasiert, das Ende eines Zeitschritts wird z.B. durch ein Nulldatenwort symbolisiert, so daß die Verarbeitung auf der nächst höheren Abstraktionsebene datengetrieben erfolgt. Wird noch berücksichtigt, daß der Topologiedatenspeicher als ein Block vorliegt (DASH), für jedes Dendritenpotential jedoch ein eigenes Speichermodul existiert (DAME), das vorverarbeitete Resultate aktiver Neuronen weiterreicht, so sind sowohl shared-memory- als auch message-passing-Komponenten vorhanden. In den Bereich der datengetriebenen bzw. kontrollgetriebenen Architekturen läßt sich auch das vorgeschlagene parallele Simulationsverfahren für PCNN einordnen. Je nachdem, ob mit lokaler verteilter Topologiespeicherung bzw. Ereignisliste oder aber mit globalen geteilten Komponenten gearbeitet wird, kann das Verfahren also als COSH- oder COME- bzw. DASH- oder DAME-Architektur klassifiziert werden.

Grundsätzlich stellt sich bei der parallelen Simulation großer PCNN erneut die Frage, ob die Simulation *zeitgetrieben* oder *ereignisgetrieben* erfolgen soll. Der wesentliche Nachteil der ereignisbasierten Verfahren liegt im erhöhten Rechen- und Speicheraufwand für die Gewährleistung chronologisch korrekter Ergebnisse, d.h. insbesondere für die Synchronisation [Fer95]. Es müssen Zustandsinformationen und Ereigniswarteschlangen zwischengespeichert werden und für die Verwaltung dieser Informationen ist erheblicher Rechenaufwand zu leisten. Im Bereich der neuronalen Netze kommt daher überwiegend die zeitgetriebene Simulation zum Einsatz [NS92][Str94][Zel94][Mis97][Res99]. Dazu ist in [Str94][Res99] ein Vergleich von zeitbasierten Verfahren, die dort mit dem Begriff *Equitemporal Iteration* bezeichnet werden, und ereignisbasierten Verfahren, die unter *Critical Event Scheduling* subsumiert werden, angestellt worden. Für die Simulation der dort untersuchten konventionellen Neuronenmodelle (siehe auch Kap. 2.3.1) wird die ereignisbasierte Simulation verworfen mit der Begründung, daß ohnehin der überwiegende Teil der Neuronen aktiv sei und alle diese Neuronen in jedem Zeitschritt Eingänge für andere Neuronen liefern, die dann jeweils neu berechnet werden müssen. Dieser Umstand resultiert aus der Eigenschaft dieser Neuronenmodelle, keine Zustände zu speichern, sondern diese in jedem Zeitschritt neu zu berechnen und diese Zustände zudem in jedem Zeitschritt an alle Nachfolgeneuronen weiterzureichen. Von den Nachfolgeneuronen kann zudem nicht festgestellt werden, ob die Zustände der Vorgänger unverändert geblieben sind, außer diese Zustände befinden sich auf dem Ruhewert.

Nach [Res99] resultiert die hohe Anzahl zu berechnender Neuronen und Verbindungen aus

- einer starken Abhängigkeit innerhalb der Netze, d.h. Änderungen können sehr große oder auch weit voneinander entfernte Netzbereiche betreffen,
- einer großen Zahl von Ereignissen, da jede Änderung eines Neuronenzustandes ein Ereignis auslöst,
- einer hohen Empfindlichkeit, d.h. kleine Änderungen können weitgehende Auswirkungen haben, und
- einer schnellen Ausbreitung von Änderungen durch die starke Vernetzung.

Aus diesen Gründen wird die zeitgetriebene Simulation eingesetzt, in der in jedem Zeitschritt alle Neuronen und alle Verbindungen des Netzes berechnet werden. Auf dieser Basis vereinfacht sich insbesondere die Verteilung der Simulationslast drastisch, da der zu erwartende Rechenaufwand allein aufgrund der Netzstruktur vorhersagbar und damit unabhängig vom Eingangsreiz und der Netzaktivität ist. Allerdings ist das Kommunikationsaufkommen maximal, so daß diese Simulationsverfahren aufgrund des schlechten Verhältnisses von Rechenaufwand zu Kommunikationsaufwand für die Parallelisierung schlecht geeignet sind. Diese Aussage wird durch die meist mäßigen Geschwindigkeitszuwächse der Verfahren auf realen Parallelrechnersystemen gestützt [Res93b][Dit94][Str94].

Im Gegensatz dazu erscheint die ereignisbasierte Simulation für PCNN sinnvoll, da diese Kriterien dort nicht oder nur vermindert zutreffen. Insbesondere wird der Aktivitätszustand eines Neurons pulscodiert übertragen. Diese Pulse werden schon aufgrund der Refraktärperiode nur selten ausgelöst. Somit ergibt sich eine weit geringere Kommunikationslast bei gleichzeitig aufgrund des komplexeren Modellneurons erhöhtem Rechenaufwand. Vor allem ergibt sich jedoch eine nur geringe Anzahl von zu berechnenden Neuronen und Verbindungen pro Zeitschritt (vergl. Kap. 3.5), da insgesamt weitaus weniger Ereignisse ausgelöst werden. Dieser Umstand rechtfertigt das Hinzufügen des - wie später noch gezeigt wird - gegenüber den Neuronenberechnungen kleinen Aufwandes für die ereignisbasierten Simulationsmethoden. Zu beachten ist, daß dieser Aufwand aufgrund der insgesamt immer noch sehr hohen Anzahl von Ereignissen möglichst gering gehalten werden muß.

Das vorgeschlagene Verfahren verwendet daher eine Mischung aus ereignisbasierten Ansätzen und einem Zeitschrittverfahren. Innerhalb der Abkling- und Erregungsphase eines Zeitschritts werden nur die Neuronen aus den Ereignislisten Spikeliste und Abklingliste verarbeitet. Die Phasen selbst und der Zeitschritt insgesamt werden jedoch synchron von allen Rechenknoten weitergeschaltet. Dieses Vorgehen erscheint sinnvoll, da ohnehin in jedem Zeitschritt eine große Anzahl von Ereignissen anfällt und daher beide Phasen des Zeitschritts berechnet werden müssen. Allerdings ist eine gute Lastverteilung für ein solches Verfahren unerläßlich, so daß in weiteren Abschnitten dieses Thema behandelt wird.

Nachdem nunmehr aus dem Bereich der drei wesentlichen Aufgabenstellungen bei der Parallelisierung sequentieller Algorithmen [Rüc96] in Form von

- *Partitionierung* des Gesamtmodells in möglichst unabhängige Teilmodelle,
- *Verteilung* der Teilmodelle auf die Rechenknoten unter Lastverteilungsaspekten und
- *Synchronisation* der Berechnungen zur Gewährleistung der globalen Kausalität

der erste Punkt in Grundzügen behandelt worden ist und zum zweiten Punkt auf die Erläuterung in weiteren Abschnitten verwiesen ist, wird nachfolgend die Thematik der Synchronisation behandelt werden, indem der gewählte Ansatz in den Kontext der Nomenklatur der verteilten ereignisbasierten Simulation gestellt wird.

Die Synchronisation paralleler Simulationsprozesse dient der Erhaltung der Kausalordnung gemäß den Vorgaben einer sequentiellen Abarbeitung der Simulation [Fer95][Rüc96], d.h. im Zusammenhang mit der PCNN-Simulation, daß der parallele Simulator eine chronologisch korrekte Erzeugung der Spike-Sequenzen garantieren muß. Eine Verletzung der Kausalordnung kann auftreten, wenn chronologisch jüngere Ereignisse vor älteren Ereignissen bearbeitet werden, obwohl sie von diesen abhängig sind. Für unabhängige Ereignisse ist ein solcher Vorgang unkritisch. Nach [Rüc96] ist ein Ereignis e_2 von e_1 kausal abhängig ($e_1 \rightarrow e_2$), wenn eine der beiden folgenden Bedingungen erfüllt ist, d.h. wenn

1. e_1 einen kleineren (älteren) Zeitstempel als e_2 aufweist und entweder
   - e_1 eine Zustandsvariable verändert, die e_2 liest, oder
   - e_1 eine Zustandsvariable liest, die e_2 verändert, oder
   - e_1 eine Zustandsvariable verändert, die auch e_2 verändert.
2. ein Ereignis e_x existiert, für das gilt $e_1 \rightarrow e_x$ und $e_x \rightarrow e_2$.

Zur Vermeidung von Kausalitätsverletzungen, also der Reaktion auf e_2 vor e_1, werden Synchronisationsverfahren benutzt, die in *konservative* und *optimistische* Verfahren unterschieden werden [Fer95][Rüc96].

Konservative Verfahren stellen sicher, daß ein Ereignis erst dann bearbeitet wird, wenn alle Ereignisse mit kleinerem (älterem) Zeitstempel abgearbeitet sind, unabhängig davon, ob das jüngere Ereignis von dem älteren kausal abhängt oder nicht. Dieses kann erreicht werden, indem die Rechenknoten wie bei dem gewählten Verfahren zur PCNN-Simulation gemeinsam die Simulationszeit fortschreiben. Diese Fortschreibung erfolgt erst dann, wenn alle Rechenknoten den Empfang und die Bearbeitung der Ereignisse des aktuellen Zeitschritts beendet haben und in Reaktion darauf die Ereignisse für den nächsten Zeitschritt erzeugt und verteilt haben. Somit wird sicher verhindert, daß in einem Zeitschritt ein Ereignis bzw. eine Ereignisbotschaft empfangen wird, die im vorherigen Zeitschritt hätte bearbeitet werden

müssen. Bei der PCNN-Simulation wird mit Hilfe von Spikes kommuniziert. Das Synchronisationsverfahren muß also gewährleisten, daß die Berechnungsknoten keine weiteren aktuellen Spikes erzeugen und das Kommunikationssystem alle Spikes des Zeitschrittes verteilt hat, bevor der Zeitschritt weitergeschaltet wird. Die Signalisierung des Zeitschrittabschlusses erfolgt im Rahmen dieser Arbeit durch eine zentrale Instanz, welche die Systemzeit fortschreibt. Dieses Vorgehen kann als *time-step* oder *time-window* Verfahren klassifiziert werden, bei denen eine feste Instanz den Zeitbereich vorgibt, nachdem nicht mehr mit dem Empfang älterer Ereignisse zu rechnen ist [Fer95]. Bei dem gewählten Verfahren ist dieser Zeitraum auf einen Zeitschritt begrenzt, so daß der Aufwand für das Verfahren minimiert wird. Bei einem an der Universität Bonn verfolgten PCNN-Simulationsansatz [GA98] werden Untersuchungen aufwendigerer Verfahren und größerer Zeitfenster angestellt, mit denen die Möglichkeiten der verteilten ereignisbasierten Simulation weiter ausgenutzt werden sollen. Im SPIKE128k-System erfolgt diese Signalisierung durch den Austausch von Null-Botschaften, was grob dem *Chandy-Misra*-Verfahren entspricht [Fer95][Rüc96]. Bei allen Verfahren muß auf die Vermeidung von deadlock-Situationen geachtet werden.

Bei der optimistischen Synchronisation wird der Zeitschritt auch dann weitergeschaltet, wenn nicht sicher ist, ob noch ältere relevante Ereignisbotschaften empfangen werden [Fer95]. Geschieht dieses dann doch, erfolgt in der Regel ein sogenannter *roll back*, der den Systemzustand vor der durch den Zeitstempel des unerwartet empfangenen Ereignisses angegebenen Zeit wiederherstellt. Es existieren aufwendige Verfahren zur Gewährleistung dieses roll back, die zudem recht speicherintensiv sind.

Konservative Synchronisationsverfahren sind nach [Fer95] besser für feinkörnig parallele Probleme geeignet, da es hier bei der optimistischen Synchronisation aufgrund der hohen Anzahl von Ereignissen zu übermäßig vielen roll back Operationen käme. Viele grobkörnig parallele Probleme profitieren hingegen von der optimistischen Synchronisation, da die Abhängigkeiten geringer sind und Schwankungen der Simulationslast im sequentiellen Anteil der Berechnungen besser ausgeglichen werden können. Pulscodierte neuronale Netze bieten sich aufgrund der sehr hohen Anzahl von Ereignissen für ein möglichst einfaches konservatives Synchronisationsverfahren an. Die aufwendigeren Ansätze aus [GA98] erscheinen für die dort untersuchten geringen Neuronenanzahlen (einige hundert) angebracht.

4.2 Kommunikation im parallelen Simulator

Neben den mit der Synchronisation und Lastverteilung einhergehenden Fragestellungen liegt ein wesentliches Problem vieler paralleler Verfahren in der Kommunikation zwischen den Rechenknoten. Wird die Kommunikation nicht genügend schnell abgewickelt, warten

die Rechenknoten auf Daten und sind an der Fortführung der Berechnungen gehindert. Während bei Lastverteilungsproblemen der langsamste Rechenknoten die Geschwindigkeit des Gesamtsystems bestimmt, wird diese bei Kommunikationsproblemen durch den größten Kommunikationsengpaß bestimmt. Bei der Entwicklung paralleler Verfahren gilt es also, die Kommunikation insgesamt so gering wie möglich zu halten und Engpässe aufzuweiten.

Zu diesem Zweck sollen noch einmal exemplarisch die ermittelten Werte für das Weitzelnetz (siehe Kap. 3.5) zusammengefaßt werden. Werden zur Kommunikation die Aktivitätswerte der relevanten Neuronen ausgetauscht, wie bei [Res99] für die Simulation konventioneller Neuronenmodelle realisiert bzw. auch im Fall der Phasenparallelität (siehe Kap. 4.1), so müßten die Daten von 16.993 Neuronen pro Zeitschritt übergeben werden. Jeder dieser Datensätze umfaßt mindestens fünf Datenworte (Dendritenpotentiale und Schwellwert) pro Neuron, aus denen der Empfänger berechnen kann, wie das Neuron auf seine Ziele einwirkt.

Demgegenüber stehen 28.269 Einzelerregungen, die ebenfalls die nötigen Modifikationen an den Neuronendaten eines Empfängerneurons erlauben. Diese Datenmenge ist geringer als die Menge der Neuronendaten der aktiven Neuronen, da eine einzelne Erregung aus weit weniger Information besteht. Eine Erregungsinformation, die im vorgeschlagenen Verfahren senderorientiert im EIB des erregenden Neurons enthalten ist, besteht prinzipiell aus einem Verbindungsgewicht und der Zieladresse des zu erregenden Neurons. Angelehnt an das biologische Vorbild kann eine solche Information als postsynaptischer Spike, d.h. als PostSpike, aufgefaßt werden. Dieser PostSpike eignet sich sehr gut zur Kommunikation, da er eine Information über sein Ziel in Form der Neuronenadresse des Zielneurons enthält. Die Kommunikation mittels PostSpikes wird in [Sch93][SHK94] für die parallele PCNN-Simulation vorgeschlagen. Die Rechenknoten ermitteln, welche Neuronen einen Spike emittieren, und bestimmen dann über eine senderorientierte Topologiespeicherung auf dem Rechenknoten die Zielneuronen und daraus die Zielrechenknoten. Die Erregungsdaten werden in Paketen für die jeweiligen Zielknoten gesammelt und über ein spezielles Kommunikationssystem versendet. Das in [Sch93][SHK94] beschriebene System wird im weiteren noch erläutert.

Die Kommunikation mittels der PostSpikes führt allerdings immer noch zu einem erheblichen Kommunikationsaufkommen. Werden hingegen die präsynaptischen PreSpikes betrachtet, die über die Netztopologie die PostSpikes auslösen, so muß nur eine Datenmenge von 453 PreSpikes pro Zeitschritt im Weitzelnetzbeispiel übertragen werden. Wird dann der jeweilige Teil der Topologiedaten nicht auf dem Rechenknoten abgelegt, auf dem das sendende präsynaptische Neuron simuliert wird, sondern auf dem Knoten, der die empfangenden postsynaptischen Ziele enthält, so kann dort aufgrund des PreSpikes ebenfalls die Modifikation der Neuronendaten vorgenommen werden. Die PreSpikes stellen somit die geringste mögliche Datenmenge dar, auf der die Kommunikation für eine parallele Simulation aufsetzen kann, und empfehlen sich damit für einen parallelen Simulator. Ein solches Vorgehen ist zudem auch mit Blick auf das biologische Vorbild plausibel, da hier die Spikes über

ein einzelnes Axon ins Zielgebiet übertragen werden und erst dort eine Auffächerung in eine große Anzahl von Synapsen erfolgt. Insofern erscheint es sinnvoll, die erzeugten PreSpikes zu solchen Rechenknoten zu versenden, die Ziele für den Spike aufweisen, und erst dort die Vervielfältigung über die Netztopologie in die PostSpikes vorzunehmen (siehe Abb. 4-4). Da sehr viele Zielneuronen auf einem Rechenknoten simuliert werden, reduziert sich die Datenmenge auf einen PreSpike pro Zielrechenknoten gegenüber den 10 bis 100 PostSpikes, die entsprechend der Konnektivität des Netzes ansonsten übertragen würden.

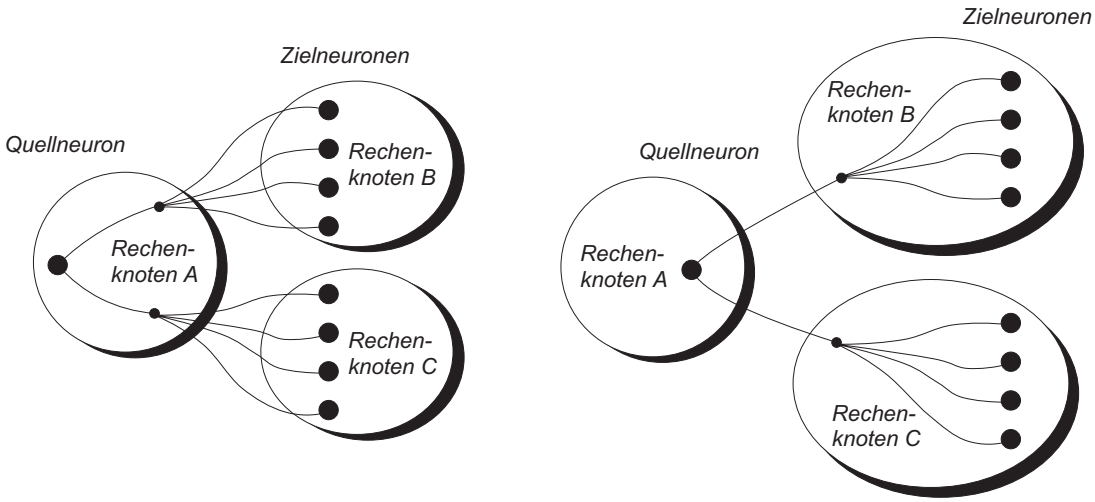

Abb. 4-4 Varianten der Spike-basierten Kommunikation

Ein wesentliches Problem der Spike-basierten Kommunikation ist die große Anzahl sehr kleiner Datenpakete, wenn jeder Spike einzeln verschickt wird. Für die meisten Kommunikationssysteme ist diese Form der Kommunikation sehr ungünstig, da jeder Datentransfer mit einem festen Grundaufwand unabhängig von der Paketlänge verbunden ist. Damit kommen Latenzzeiten zustande, die sich zu den reinen Übertragungszeiten addieren. Viele Datenpakete bedeuten einen großen Anteil von Latenzzeiten, so daß die Übertragungskapazität des Kommunikationskanals sinkt. Umgangen werden kann dieses Problem durch das Sammeln der Spikes zu größeren Paketen. Allerdings verlangt das gewählte konservative Synchronisationsverfahren, daß alle Spikes bis zum Zeitschrittende an ihren Zielen sind. Zudem treten bei einer PreSpike-basierten Kommunikation sehr wenige Spikes auf, so daß sich insgesamt sehr kleine Pakete ergeben. Ein Sammeln der Spikes führt außerdem zu Lastverteilungsproblemen, da erst nach Empfang des gesamten Pakets vom Zielrechenknoten reagiert werden kann und sich somit dort Wartezeiten ergeben können. Diese Problematik wird im Rahmen der in Kap. 5 vorgestellten Softwaresimulationen eingehender behandelt.

Die Kommunikation allein auf Basis gleichförmiger PreSpikes eröffnet allerdings Möglichkeiten, den Transfer schaltungstechnisch abzubilden, da eine einfache und einheitliche Routing-Hardware möglich wird. Im Gegensatz zum PostSpike enthält der PreSpike jedoch

keine Information über sein Ziel. Um trotzdem mittels PreSpikes zu kommunizieren, bieten sich verschiedene Ansätze an, die im weiteren im Rahmen der Softwaresimulationen oder des vorgestellten Hardwarekonzepts untersucht werden.

- Alle PreSpikes können zu einer zentralen Spikeliste übertragen werden, mit Hilfe derer dann über eine zentrale Topologiespeicherung die postsynaptischen Ziele ermittelt werden. Dieses Vorgehen ist in Abb. 4-1 in Form der gemeinsamen Spikeliste und Netztopologie dargestellt. Die gemeinsam genutzten Komponenten lassen Engpässe beim Zugriff erwarten. Im folgenden Kapitel wird eine Möglichkeit geschildert, diesen Engpaß zu reduzieren.

- Die PreSpikes werden in verteilten lokalen Spikelisten dort gesammelt, wo ihre Zielneuronen simuliert werden. Dort befinden sich auch die benötigten Topologiedaten. Dabei tritt das Problem auf, die PreSpikes zu diesen Listen zu übertragen, da der Rechenknoten des sendenden Neurons zunächst keine Information besitzt, wo das sendende Neuron Ziele hat. Eine Möglichkeit der Lösung ist das Versenden des PreSpikes an alle Rechenknoten. Die Zielrechenknoten wissen, ob sie Zielneuronen für diesen PreSpike beherbergen, und verwerfen den PreSpike andernfalls. Dieses Vorgehen ist in [DDW98] als *Address-Event-Bus* vorgeschlagen und schaltungstechnisch in Form eines einzigen Busses implementiert, an dem alle Rechenknoten Teilnehmer sind. Die Spikes werden in Anlehnung an [Wat94] als *Address-Events* bezeichnet. Im Rahmen der nachfolgend dargestellten Softwaresimulationen [Cib99] ist dieses Vorgehen untersucht worden. Es läßt sich absehen, daß es mit zunehmender Rechenknotenzahl schlecht skaliert.

- Wenn der Rechenknoten des sendenden Neurons die Information hat, an welchen Rechenknoten er den PreSpike versenden muß, kann eine gezielte Kommunikation erfolgen, die bei einer schaltungstechnischen Implementierung den Einsatz von Kommunikationsnetzen erlaubt und die Broadcast-Übertragung ersetzt. Dazu muß der Rechenknoten des Zielneurons jedoch nicht die komplette Netztopologie bei sich speichern, sondern es genügt eine Liste mit Zielrechenknoten für ggf. jedes lokal simulierte Neuron. Diese Liste enthält weit weniger Einträge als ein EIB eines Neurons (siehe Abb. 4-4), da viele Zielneuronen auf dem gleichen Zielrechenknoten beheimatet sind. Die Länge der Listen wird dabei durch die Verteilung der Neuronen auf die Rechenknoten beeinflußt. Je nach Organisation der Verteilung kann die gleiche *Zielliste* für eine ganze Gruppe von Quellneuronen benutzt werden. Für eine in Hardware abgebildete Übertragung der PreSpikes kann diese Information über den Zielknoten in normierter Form in das PreSpike-Datenformat aufgenommen werden und so zum Routing dienen. Es wird dann für jedes Ziel eine eigener PreSpike erzeugt. Die Anzahl bleibt dabei jedoch weit hinter der Zahl von PostSpikes zurück, die später beim Zielknoten erzeugt wird, und kann durch eine geschickte Netzpartitionierung gering gehalten werden. Auch zu diesen Verfahren werden nachfolgend Simulationen vorgestellt und Implementie-

rungskonzepte präsentiert.

4.3 Methoden der Netztopologiedarstellung

Für die Kommunikation und damit für den gesamten Ablauf im parallelen Simulator ist die Form der Netztopologiedarstellung und insbesondere der Ort ihrer Speicherung von wesentlicher Relevanz, da die Netztopologie - mit diesem Begriff wird die Verknüpfungsstruktur des Netzes bezeichnet - die Kommunikationswege im zu simulierenden neuronalen Netz beschreibt. Als besonders effizient für die sequentielle Simulation hat sich die *dynamische senderorientierte Topologiespeicherung* (siehe Kap. 3.4.1) erwiesen, so daß Möglichkeiten gesucht sind, diese Form der Topologiespeicherung an den parallelen Simulator anzupassen.

Zunächst einmal kann dazu die *globale Topologiespeicherung* (Abb. 4-5) in einem gemeinsamen Speicher (shared memory), die der Topologiespeicherung des sequentiellen Simulators mit dem Unterschied des Zugriffs durch mehrere Recheneinheiten entspricht, von einer *lokalen Topologiespeicherung* auf den Rechenknoten (Abb. 4-6) unterschieden werden.

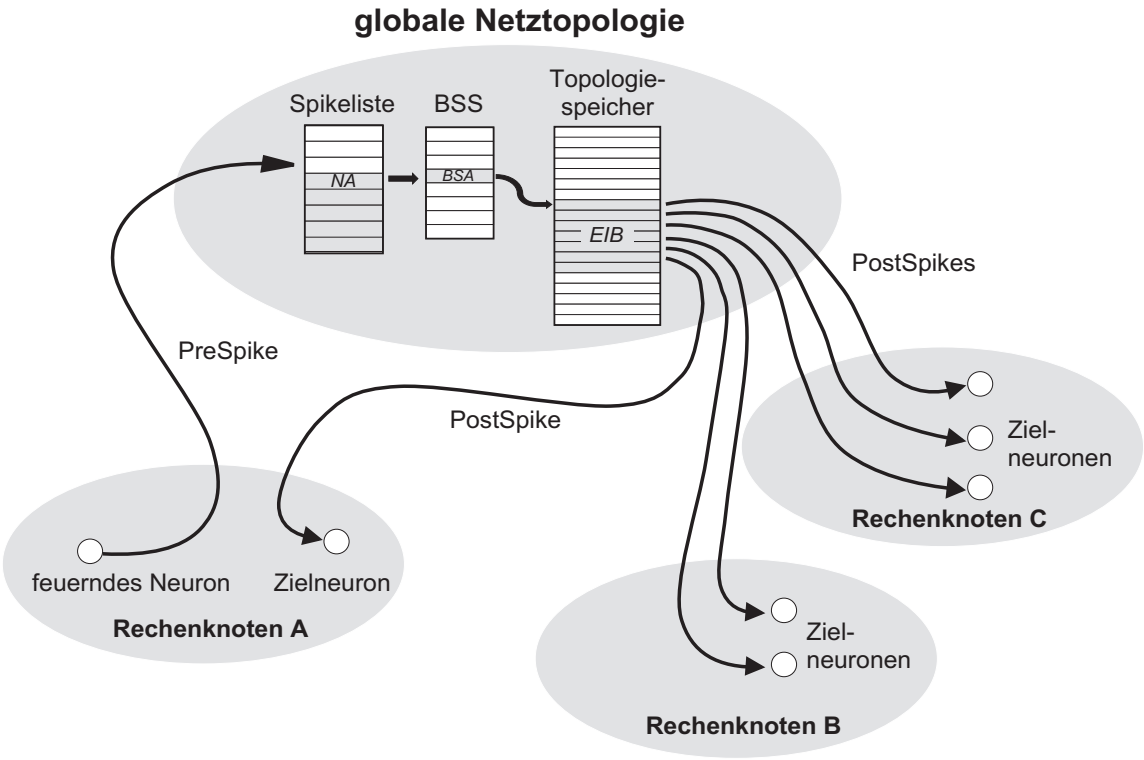

Abb. 4-5 Simulation mit globaler Netztopologiespeicherung

Die Simulation mit einer zentralen Ablage der Netzstruktur und gemeinsamem Zugriff der Rechenknoten auf diese globale Ressource entspricht dem in Abb. 4-1 dargestellten Grundverfahren. Dazu werden alle in der Abklingphase erzeugten PreSpikes in einer globalen

Spikeliste von einer speziellen Einheit zur Verwaltung des gemeinsam genutzten *Topologiespeichersubsystems* (shared memory) gesammelt. Damit handelt es sich zugleich um ein Verfahren mit einer zentralen Ereigniswarteschlange. Von der Topologiespeichereinheit werden in der Erregungsphase mit den gesammelten PreSpikes über die EIBs die PostSpikes ermittelt und diese werden an die Zielrecheneinheiten zur Verarbeitung übertragen.

Dieses Vorgehen bietet im wesentlichen zwei Vorteile. Zum einen benötigen die Rechenknoten keinen lokalen Topologiespeicher, der auch bei einer Aufteilung der gesamten Topologie immer noch signifikant größer als der lokale Neuronenspeicher wäre und somit möglicherweise die lokalen Ressourcen des Rechenknotens übersteigt. Insbesondere bei einer schaltungstechnischen Realisierung läßt sich ein einziger zentraler Topologiespeicherblock auf Basis moderner DRAM-Technik mit wesentlich weniger Aufwand realisieren, als viele kleinere lokale Speichereinheiten. Ein weiterer Vorteil ergibt sich, wenn auf der Netztopologie Lernverfahren eingesetzt werden sollen [Sch95-00][SH99][Sch00][SSW00]. Diese benötigen zusätzlichen Speicheraufwand und sind auf den Zugriff auf alle Verbindungen eines Neurons angewiesen, so daß bei einer lokalen Topologiespeicherung zusätzlicher Kommunikationsaufwand und Speicherbedarf für die Vervielfältigung der Lerneinheit entstünde. Eine zentrale Lerneinheit läßt sich hingegen gut mit einem zentralen Topologiespeicher verbinden [Sch00][SSW00].

Das Verfahren birgt jedoch prinzipiell den Nachteil des Zugriffs paralleler Rechenknoten auf eine gemeinsame Ressource in sich. Damit ist der typische Fall eines Engpasses in einem parallelen System gegeben. Dabei stellt der Eintrag der PreSpikes in die gemeinsame Spikeliste das kleinere Problem dar, da die PreSpikes nur eine sehr geringe Datenmenge bilden. Gegenüber dem Kommunikationsverfahren mit verteilten lokalen Spikelisten ist der Aufwand sogar geringer, da ein PreSpike nur in eine Spikeliste eingetragen und damit nur einmal übertragen werden muß. Gravierender erscheint das zentrale Suchen und Lesen der EIBs sowie insbesondere der Transfer der großen Menge von PostSpikes zu den parallelen Einheiten.

Dieser Engpaß kann nur umgangen werden, wenn die EIBs oder zumindest Teile der EIBs lokal auf den Rechenknoten gelagert werden. Abgesehen von der Frage, wie sie in begrenzten lokalen Speichern untergebracht werden sollen, ergeben sich zwei Möglichkeiten, die EIBs zu verteilen. Zum einen kann der EIB auf dem Rechenknoten des Neurons gespeichert werden, das bezüglich dieses EIBs präsynaptisch ist, was der senderorientierten Zusammensetzung des EIBs aus den Verbindungen eines sendenden Neurons zu seinen Nachfolgern entgegenkommen würde. Dieser in [Sch93][SHK94] untersuchte Weg führt jedoch zu einer Kommunikation mit PostSpikes und löst das Problem somit nicht. Werden die EIBs jedoch auf den Rechenknoten abgelegt, auf denen die enthaltenen postsynaptischen Zielneuronen beheimatet sind, so kann mit PreSpikes kommuniziert werden. Dazu muß der EIB eines jeden Sendeneurons bezüglich der enthaltenen Nachfolger aufgespalten werden, da diese auf

verschiedenen Rechenknoten beheimatet sein können.

Diese Aufspaltung der EIBs und ihre Einlagerung auf den Rechenknoten kann *statisch* vor Beginn der Simulation erfolgen oder *dynamisch* während der Simulation. Zur *dynamischen Reallokation* der Netztopologiedaten kann sehr gut das in Kap. 3.4.3 beschriebene *EIB-Caching-Verfahren* zum Einsatz kommen. Dieser EIB-Cache ergänzt dann das Verfahren mit globaler Topologiespeicherung (siehe Abb. 4-5). Aufgrund der hohen Lokalität der Netzaktivität kann wie in Kap. 3.4.3 gezeigt schon mit einem kleinen Cache-Speicher eine hohe Trefferrate erreicht werden, so daß der Transfer für die dort gefundenen PostSpikes entfällt. Die Einlagerung eines EIB-Teils in den Cache macht natürlich nur Sinn, wenn das feuernde Neuron und die in diesem EIB-Teil enthaltenen Zielneuronen auf dem gleichen Rechenknoten liegen. Ist das der Fall, so kann neben der Abklingphase auch die Erregungsphase für diese Verbindungen komplett lokal auf dem Rechenknoten ablaufen. Liegen Zielneuronen auf anderen Rechenknoten, d.h. werden Verbindungen durch das Verteilen der Neuronen auf die Rechenknoten geschnitten, so muß für diese Verbindungen der Umweg über den gemeinsamen Topologiespeicher gewählt werden. Es ist daher Aufgabe eines guten Partitionierungsverfahrens für das neuronale Netz, die Anzahl dieser Schnitte gering zu halten.

Ist auf den Rechenknoten genügend Speicher vorhanden und wird auf den Einsatz eines Lernverfahrens verzichtet, so kann die *komplette Topologie* vor Beginn der Simulation *statisch* auf die Rechenknoten verteilt werden. Dazu werden die EIBs aufgespalten und auf die

Rechenknoten verteilt, auf denen die in den EIB-Zeilen enthaltenen Ziele beheimatet sind.

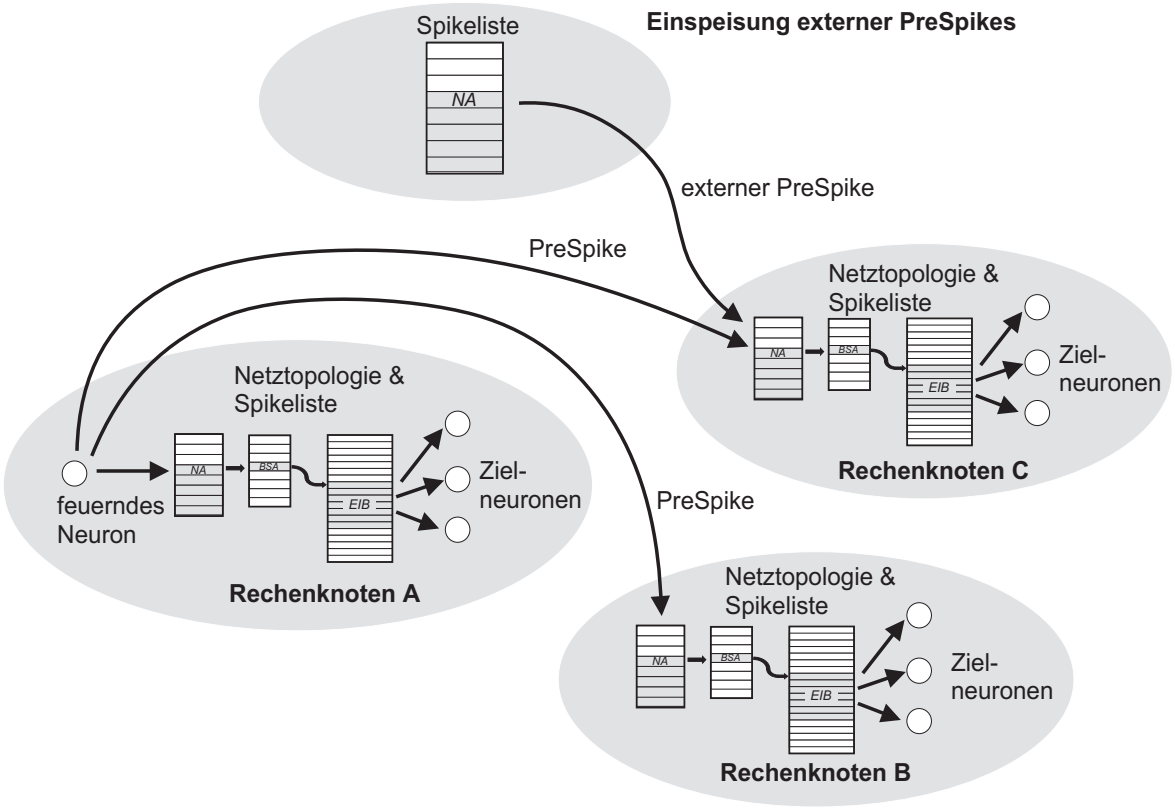

Abb. 4-6 Simulation mit verteilter lokaler Netztopologiespeicherung

Zusammen mit einer lokalen Speicherung der verteilten Spikeliste kann das rein PreSpike-basierte Kommunikationsverfahren für verteilte Ereigniswarteschlangen zum Einsatz kommen, d.h. während der Abklingphase werden die erzeugten PreSpikes an die Zielrechenknoten übertragen und dort in lokale Spikelisten eingetragen. Die Übertragung kann im Broadcast an alle Rechenknoten oder gezielt über Ziellisten erfolgen. Das Verfahren benötigt neben dem lokalen EIB-Speicher einen lokalen Block-Start-Speicher BSS, der Einträge für alle Neuronen im Netz enthält, da der Rechenknoten potentiell von allen Neuronen im Netz PreSpikes empfangen kann. Hat ein Neuron keine Nachfolger auf einem Rechenknoten, so enthält der dortige BSS unter der Neuronenadresse dieses Neurons einen Null-Eintrag. Werden PreSpikes im Broadcast übertragen, so kann der Rechenknoten mit Hilfe dieses BSS-Eintrags einen für ihn irrelevanten PreSpike verwerfen. Nachteilig ist, daß die Größe dieses lokalen BSS nicht mit der Anzahl der Neuronen auf dem Rechenknoten sondern der Anzahl aller Neuronen wächst, während der Bedarf aller anderen zur Simulation notwendigen Speicher sich an der Anzahl der lokalen Elemente orientiert. Durch geschickte Vergabe der Neuronenadressen und Zusammenfassen der Neuronen, die Ziele auf einem Knoten haben, läßt sich der Aufwand für den BSS verringern, wobei jedoch Freiheitsgrade bei der Netzpartitionierung verloren gehen.

Die Verteilung der gesamten Netztopologie auf die Rechenknoten erfordert erhebliche lokale Speicherressourcen bei den Rechenknoten, die möglicherweise nicht zur Verfügung stehen, so daß der Bedarf nach einer kompakteren Topologiedarstellung besteht. Dazu bietet sich ein Ansatz an, der die hohe Regularität (siehe Kap. 2.4.4) der betrachteten Bildverarbeitungs-PCNN ausnutzt.

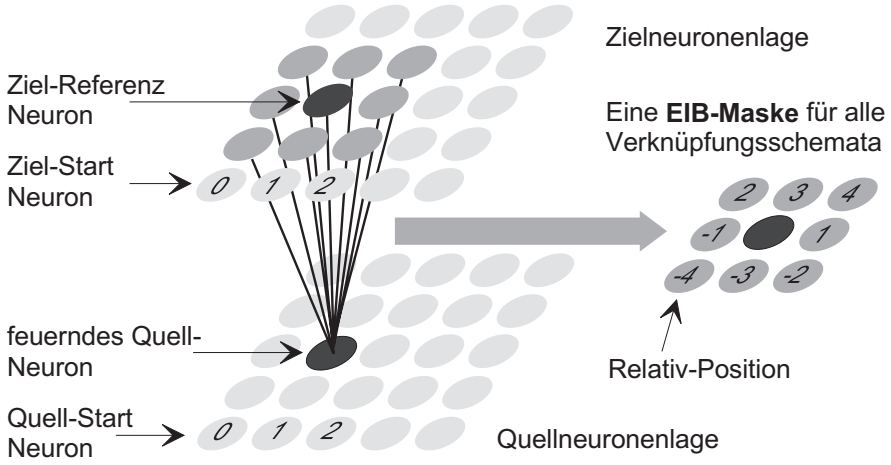

Abb. 4-7 Topologierepräsentation mittels regulärer EIB-Masken (rc)

Während im Fall der verteilten Topologiedaten für jedes Neuron ein eigener EIB auf den Rechenknoten gespeichert wird und im Falle des EIB-Caches für einige Neuronen die häufig benötigten EIBs abgelegt werden, kann unter Ausnutzung der gleichförmigen Netzstruktur auch ein einziger EIB für eine ganze Anzahl von Neuronen verwendet werden. Sind diese Neuronen nämlich mittels des gleichen Verknüpfungsschemas mit ihren Nachfolgern verbunden, kann statt des konkreten EIBs auch eine *EIB-Maske* (Abb. 4-7) gespeichert werden. Im weiteren wird eine solche EIB-Maske auch als *rc-EIB* (regular connection) bezeichnet. Aus den dort gespeicherten Relativpositionen der Ziele lassen sich für das konkrete Quellneuron die konkreten Ziele berechnen. Bei einem lagenweise aufgebauten Bildverarbeitungsnetz ist eine solche Maske für alle Neuronen einer Lage gültig. Das Konzept paßt sich nahtlos in die senderorientierte Topologiespeicherung ein, da einzig die BSS-Zeiger aller Neuronen einer Lage statt auf unterschiedliche EIBs auf den gleichen rc-EIB zeigen müssen.

Um die Berechnung der Absolutpositionen der Zielneuronen aus den Relativpositionen innerhalb des rc-EIBs vornehmen zu können (siehe Abb. 4-8), bedarf es einiger Zusatzinformationen, die zusätzlich im EIB abgelegt werden müssen [Rah00][Weg00]. Zunächst einmal muß aus der Neuronenadresse eines feuernden Quellneurons NA_Q die Position dieses Neurons innerhalb der Lage (*sx*,*sy*) berechnet werden. Dazu werden der Adreß-Offset der Lage $StartNA_Q$ sowie die Abmessungen der Lage in x-Richtung N_Q und in y-Richtung M_Q benötigt. Mit diesen Werten läßt sich dann gemäß

$$sx = (NA_Q - StartNA_Q) mod N_Q$$
$$sy = (NA_Q - StartNA_Q) / N_Q \quad \text{(Gl. 3.30)}$$

die Position (*sx*,*sy*) des Neurons berechnen. Die Koordinate *sx* entsteht durch eine Modulo-Operation, d.h. sie ist der Rest der ganzzahligen Division durch die x-Ausdehnung der Lage. Entsprechend ist *sy* das Ergebnis dieser ganzzahligen Division.

Mit den Relativkoordinaten (*x*,*y*) des Ziels aus dem entsprechenden Eintrag in der EIB-Maske ergeben sich dann die Zielkoordinaten (*tx*,*ty*) gemäß:

$$tx = sx + x$$
$$ty = sy + y \quad \text{(Gl. 3.31)}$$

Diese Koordinaten können den Umfang der Ziellage überschreiten, so daß überprüft werden muß, ob

$$0 \le tx < N_Z$$
$$0 \le ty < M_Z \quad \text{(Gl. 3.32)}$$

gilt. Ist diese Bedingung erfüllt, so kann mit

$$NA_Z = StartNA_Z + ty \cdot N_Z + tx \quad \text{(Gl. 3.33)}$$

die Adresse des Zielneurons berechnet werden. Im rc-EIB ersetzen die Relativkoordinaten (*x*,*y*) gerade die Neuronenadresse des Zielneurons im normalen (nrc) EIB. Die weiteren In-

formationen über Quell- und Ziel-Lage sind im rc-EIB-Kopf abgelegt.

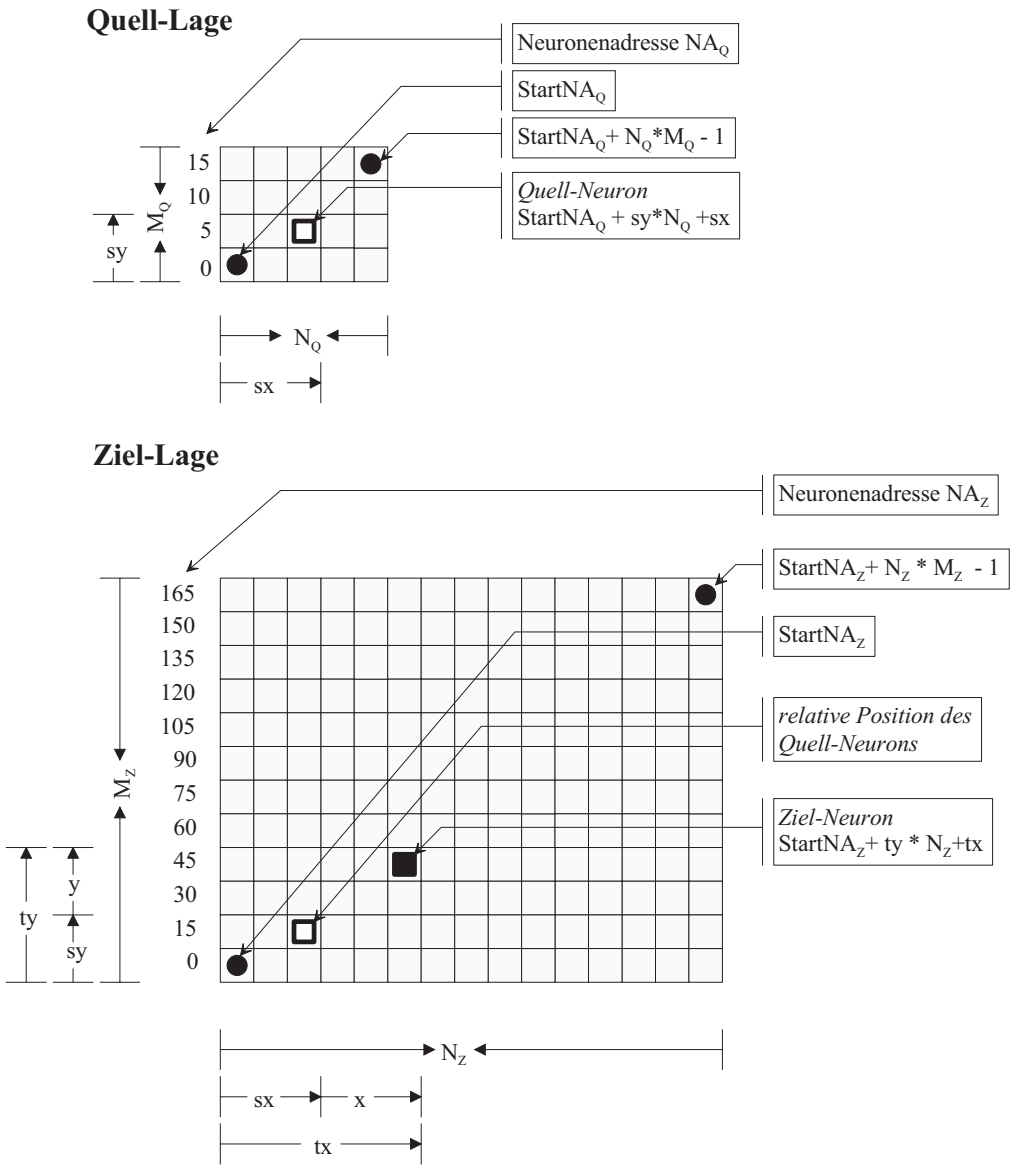

Abb. 4-8 Berechnungen zur Verarbeitung regulärer Strukturen (rc-EIB)

Die Verwendung von rc-EIBs kann anstelle oder ergänzend zu einer vollständig auf die Rechenknoten verteilten Netztopologie verwendet werden, am Kommunikationsverfahren und an allen anderen Berechnungen ändert sich dabei nichts. Vorbedingung ist jedoch, daß ganze Neuronenlagen mittels des gleichen Verknüpfungsschemas mit anderen Lagen verknüpft sind. Außerdem müssen die Lagen zur Anwendung der geschilderten einfachen Berechnungen voll besetzt und rechteckig sein. Es ist jedoch möglich, nicht vollbesetzte Neuronenlagen auf vollbesetzte zurückzuführen, wenn neben einigen anderen Randbedingungen in jeder Zeile der Lage gleich viele Neuronen enthalten sind.

Alternativ zu den EIB-Masken kann eine solche reguläre Verknüpfungsstruktur auch in Form eines mathematischen Ausdrucks gespeichert werden, mit dem über die Koordinaten des Quellneurons das Zielneuron berechnet wird [REJ97][RJK97]. Eine solche Maskenfunktion läßt sich möglicherweise noch kompakter speichern. Zudem werden, wie noch gezeigt wird, in der verwendeten Netzbeschreibungssprache sowohl in dieser Arbeit als auch in [REJ97][RJK97] mathematische Funktionen zur Berechnung der Verknüpfungsstrukturen benutzt. Auf konventionellen Mikroprozessoren ergibt sich dabei jedoch ein sicherlich größerer Berechnungsaufwand als bei der Verwendung der rc-EIBs, bei einer speziellen Hardware ist die Vielfalt der möglichen Verknüpfungsmasken durch die interpretierbaren Funktionen der Implementierung beschränkt.

Zusammenfassend werden im weiteren drei Formen der senderorientierten Topologiespeicherung im parallelen Simulator betrachtet, die sich alle in das BSS-EIB-Konzept aus Kap. 3.4.1 einfügen.

- Die *Topologiespeicherung in einem gemeinsamen EIB-Speicher* für alle Rechenknoten benutzt teilweise eine PostSpike-basierte Kommunikation mit zentraler Spikeliste. Durch den Einsatz des *EIB-Caches* wird ein Teil der Topologiebehandlung lokal abgewickelt.
- Die *verteilte Speicherung der Topologie* auf den Rechenknoten geht mit einer PreSpike-basierten Kommunikation und verteilten lokalen Spikelisten einher. Sie stellt jedoch sehr hohe Anforderungen an den lokalen Speicher und erlaubt nicht ohne weiteres die Integration der in [SH99][SSW00][Sch00] vorgestellten Lernverfahren für PCNN.
- Die *Speicherung regulärer Netztopologien (rc)* mit Hilfe von rc-EIBs erlaubt die platzsparende Umsetzung der verteilten Topologiespeicherung. Sie ist jedoch auf bestimmte Netzstrukturen angewiesen und erlaubt ebenfalls nicht die Integration der Lernverfahren.

Neben der senderorientierten Topologiespeicherung ist für die Lernverfahren auch die Bereitstellung einer inversen empfängerorientierten Topologiedarstellung hilfreich. Diese kann in Form eines Zeigerspeichers [SSW00][Sch00][Ull00] realisiert werden, der zu jedem Empfängerneuron einen inversen EIB (iEIB) enthält. In diesem iEIB verweisen die Einträge auf die entsprechenden EIB-Zeilen der Verbindungen der Vorgängerneuronen zu diesem Empfängerneuron. Zudem ist die Neuronenadresse des Vorgängerneurons enthalten, so daß über den iEIB ausgehend von einem Empfängerneuron auf die Vorgängerneuronen und die Verbindungsdaten zugegriffen werden kann. Kombiniert mit dem Speichersubsystem der zentralen Topologiespeicherung können während der Abklingphase, in der ansonsten keine Zugriffe erforderlich sind, Lern- und Verlernvorgänge von einer Lerneinheit durchgeführt werden. Der Aufbau eines solchen Speichersubsystems ist in Kap. 6.4 sowie in [Ull00] dargestellt, die Lernverfahren in der Dissertation [Sch00].

4.4 Partitionierung neuronaler Netze

Das vorgestellte Simulationsverfahren und insbesondere die Abwandlungen bezüglich der Topologiespeicherung und der Kommunikation haben gemeinsam, daß die Verteilung des neuronalen Netzes erheblichen Einfluß auf die erreichbare Leistung hat. Im Fall der PreSpike-basierten Kommunikation wird die Zahl der Rechenknoten, zu denen ein PreSpike verschickt werden muß, durch die Zahl der geschnittenen Netzverbindungen bestimmt. Die Trefferrate des lokalen EIB-Caches ist ebenfalls vom Verbindungsschnitt abhängig. Die Verteilung der Netzaktivität auf die Rechenknoten ist zudem entscheidend für die Verteilung der Rechenlast. Aus dieser Problemstellung entsteht die Forderung nach einer geeigneten Netzpartitionierung. Dabei ist von zwei Optimierungszielen auszugehen, nämlich der *Rechenlastverteilung* durch gute Verteilung der aktiven Neuronen und der *Kommunikationslastreduktion* durch möglichst wenige geschnittene Netzverbindungen.

In [Res99] wird die Verteilung eines neuronalen Netzes auf die Partitionierung von Random-Graphen zurückgeführt. Solche Graphen bestehen aus Knoten, die den Neuronen des neuronalen Netzes entsprechen, sowie Kanten, die als Synapsen interpretiert werden können. Ausgehend von der Annahme, daß eine gute Partitionierung für einen zufälligen Graphen allgemein für Graphen gute Ergebnisse erzielt, werden die Untersuchungen in [Res99] dementsprechend eingeschränkt. Dabei gilt jedoch die aus dem Simulationsverfahren resultierende Annahme, daß alle Neuronen und alle Verbindungen immer aktiv sind.

Im Fall der Bildverarbeitungs-PCNN ist diese Annahme, wie in Kap. 3 gezeigt, gerade nicht sinnvoll, es soll im Gegenteil ausgenutzt werden, daß nur sehr wenige Netzelemente aktiv sind. Welche Teile des Netzes aktiv sind und somit berechnet werden müssen, hängt zum einen vom Aufbau und zum anderen vom Eingangsreiz des Netzes ab. Damit ist die Simulationslast und auch die optimale Verteilung des Netzes auf die Rechenknoten von diesen beiden Größen abhängig.

Da das zu untersuchende bzw. zu simulierende Netz a priori in Form einer Netzbeschreibung bekannt ist, kann eine Netzpartitionierung auf diese Parameter zugreifen. Dabei erscheint es sinnvoll, neben der reinen Netzstruktur auch Wissen über die Bedeutung der Netzkomponenten in die Partitionierung einzubeziehen. Da im Rahmen dieser Arbeit große Bildverarbeitungs-PCNN betrachtet werden, können die bekannten Eigenschaften neuronaler Sehsysteme zur Verteilung genutzt werden. Dabei handelt es sich insbesondere um die folgenden Charakteristika (siehe auch Kap. 2.4.4).

- Die Neuronen in Bildverarbeitungs-PCNN sind in Lagen organisiert, wobei jeder Lage eine spezielle Funktion zukommt. Diese Funktion äußert sich darin, daß alle Neuronen der Lage vorzugsweise auf bestimmte Reiztypen reagieren.

- Es liegt eine topographische Organisation vor, d.h. die Position eines Neurons in der Lage ist mit einer Position bzw. einem Bildbereich im präsentierten Eingangsbild korreliert. Das Neuron reagiert auf Reize in diesem Bildbereich, dem rezeptiven Feld des Neurons. Benachbarte Neuronen reagieren auf Reize in benachbarten Bildbereichen.
- Neuronen sind überwiegend mit Neuronen des eigenen oder benachbarter rezeptiver Felder verbunden, d.h. die Verknüpfungsoperatoren sind lokale Operatoren.

Der dem Netz zugeführte Eingangsreiz, der üblicherweise in Form eines Bildes oder einer aus einem Bild erzeugten Spikefolge vorliegt, ist jedoch zunächst unbekannt. Im Falle einer Bildfolgenanalyse ändert sich der Eingangsreiz sogar im Laufe der Simulation. Zudem führt der Reiz nur in solchen Neuronenlagen zu einer Reaktion, die auf die enthaltenen Merkmale reagieren, und diese Reaktion findet nur bei den Neuronen der Lage statt, deren rezeptives Feld von dem Reiz berührt wird. Damit ist die Lastverteilung a priori unbekannt, d.h. es muß eine Netzpartitionierung gefunden werden, die bei allen Reizen eine gute Verteilung gewährleistet. Eingangsbilder variieren also in Form einer unterschiedlichen örtlichen Verteilung der Reize und einer unterschiedlichen Charakteristik dieser Reize.

Zur Netzverteilung lassen sich grundsätzlich zwei Ansätze verwenden [MDP96]. Vor der Simulation kann *statisch* eine Netzpartitionierung erzeugt und während der Simulation beibehalten werden. Alternativ kann eine Verteilung auch im Laufe der Simulation *dynamisch* durch Umverteilung an geänderte Lastverteilungen angepaßt werden.

Die dynamische Lastverteilung läßt sich im Fall des gewählten parallelen PCNN-Simulationsverfahrens durch den Austausch von Neuronen zwischen den Rechenknoten realisieren. Einzelne Neuronen stellen feingranulare Lastkomponenten dar, die sich für eine Umverteilung besonders gut eignen [MDP96]. Eine dynamische Lastverteilung während der Simulation führt jedoch zu zusätzlicher Kommunikation, die auch bei der PCNN-Simulation weiterhin ein kritischer Faktor ist. Zudem fügt sich ein solcher Austausch nicht in die Spikebasierten Protokolle des gewählten Verfahrens ein, so daß auch hier Zusatzaufwand entstünde. Neben diesen Problemen ergibt sich die Frage nach den Kriterien für eine Umverteilung eines Neurons auf einen anderen Rechenknoten unter Berücksichtigung der möglicherweise konkurrierenden Ziele der Lastverteilung und Kommunikationsreduktion. Insbesondere die Ermittlung des Einflusses einer Umverteilung auf den Verbindungsschnitt bedeutet erheblichen Aufwand und erfordert gegebenfalls Zugriffe auf den gemeinsamen Topologiespeicher. Da der Aufwand für eine dynamische Netzverteilung während der Simulation insbesondere dem Entwurfsziel einer einfachen parallelen Spezialhardware entgegensteht, wird der Ansatz im Rahmen dieser Arbeit nicht weiter verfolgt.

Mittels einer statischen Verteilung des Netzes soll stattdessen versucht werden, eine genügend gute Lastverteilung bei geringer Kommunikation zu erreichen, welche die angestrebte Leistung für eine parallele Simulation auch ohne dynamische Lastverteilung sichert.

Im Zusammenhang mit der statischen Lastverteilung wird das *Mapping* eines Netzes oder eines Graphen, der das Netz beschreibt, auf einen Hardware-Graphen von einer *Netzpartitionierung* ohne Berücksichtigung des Hardware-Graphen unterschieden [MDP96]. Der Hardware-Graph ist dabei die Darstellung eines parallelen Simulationssystems und seiner Kommunikationsstruktur durch eine Menge von gewichteten Knoten und Kanten. Da im Rahmen der vorliegenden Arbeit neben der Konzeption einer parallelen Spezialhardware Simulationen auf anderen parallelen Plattformen durchgeführt werden sollten, wurde auf die Verwendung eines speziellen Hardware-Graphen verzichtet, womit sich die Verteilungsaufgabe auf ein Partitionierungsproblem reduziert. Bei der Wahl einer Partitionierungsstrategie werden stattdessen die verschiedenen zugrundeliegenden Simulationsverfahren (siehe Kap. 4.3) berücksichtigt sowie bestimmte Obergrenzen für die Auslastung der Knoten.

Bei der Partitionierung werden globale Methoden von lokalen Methoden unterschieden. *Globale Methoden* erstellen eine erste Partitionierung auf Basis eines unpartitionierten Graphen, *lokale Methoden* verbessern vorgegebene Partitionierungen [MDP96]. Für die PCNN-Simulation sind die globalen Methoden relevant, da das Netz als monolitischer Block ohne Verteilungsvorgaben vorliegt. Die Methoden können dabei in verschiedene Klassen unterschieden werden [MDP96].

- *Einfache Methoden* versuchen ohne Beachtung des Verbindungsschnittes gleichgroße Knotenmengen zu bilden. Im Rahmen der PCNN-Simulation interessant ist dabei die Modulo-Verteilung der Neuronen auf die Rechenknoten. Dieser Ansatz wurde auch in [RJK95][RJK97][REJ97][JRS98] an der TU Berlin verfolgt. Dabei können einzelne Neuronen oder Neuronengruppen alternierend verteilt werden, wobei über die Vernetzungsstruktur keine Informationen vorliegen müssen. Es läßt sich erwarten, daß diese Methode zu einer guten Rechenlastverteilung führt, da Bildareale mit hoher Aktivität aufgebrochen und verteilt werden. Gleiches gilt für Lagen, die stärker als andere Lagen auf einen speziellen Reiz reagieren. Gleichzeitig ist absehbar, daß der Verbindungsschnitt und damit die Kommunikation maximal werden.

- *Geometrische Methoden* berücksichtigen die Positionen der Neuronen im Netz bzw. in der Neuronenlage und beziehen damit Informationen über die Netztopologie mit ein. Es zeigen sich zwei grundsätzliche Ansatzpunkte. Zum einen lassen sich vollständige Neuronenlagen auf gleiche Rechenknoten verteilen, was einem *horizontalen Schnitt* durch das Netz entspricht. Dabei besteht die Gefahr, daß Lagen, die stark auf einen Reiz reagieren, gehäuft auf einen Rechenknoten verteilt werden, so daß die Last ungleichmäßig verteilt wird. Zudem kann ein hoher Verbindungsschnitt auftreten, wenn Nachfolger- oder Vorgängerlagen auf andere Rechenknoten verteilt werden. Für diesen Schnitt ist das Verhältnis von Verbindungen innerhalb der Lage (sogenannte Linking-Verbindungen) zu solchen zwischen den Lagen relevant. Die andere Möglichkeit besteht in *vertikalen Schnitten*, d.h. aus den Lagen werden Neuronengruppen mit dem gleichen

rezeptiven Feld auf die gleichen Rechenknoten verteilt. Diese Methode kann auch als das Ausschneiden von Kacheln aus den Neuronenlagen bzw. von Säulen aus den übereinandergelegten Lagen aufgefaßt werden. Werden die Kacheln genügend groß gewählt, steigt die Wahrscheinlichkeit, nur wenige Linking-Verbindungen zu schneiden, da die Verknüpfungsschemata lokale Operatoren sind. Allerdings werden dadurch mit höherer Wahrscheinlichkeit auch Aktivitätsschwerpunkte in der Lage nicht aufgebrochen, so daß sich wieder die Konkurrenz von Verbindungsschnitt und Rechenlastverteilung zeigt. Horizontale und vertikale Methoden sind auch in [MSP97] an der Universität Marburg untersucht worden.

- Die *Breitensuche* berücksichtigt im Gegensatz zu den vorgenannten Methoden die Verknüpfungsstruktur des Netzes zur Auswahl der Neuronen einer Partition. Dazu werden zu bestimmten Neuronen alle Nachfolgeneuronen bis zu einer gewissen Tiefe gesucht und auf die gleiche Partition gelegt. Es ist zu erwarten, daß so ein geringerer Verknüpfungsschnitt erreicht wird, obwohl sich durch die Begrenzung der Suchtiefe Beispiele konstruieren lassen, die zu höherem Schnitt führen [Cib99]. Die Verteilung solcher zusammenhängenden Neuronengebiete führt allerdings mit hoher Wahrscheinlichkeit auch zur Verteilung ganzer Aktivitätsschwerpunkte auf einzelne Rechenknoten und so zu einer schlechten Rechenlastverteilung. Durch Auswahl der Startneuronen läßt sich eine Kombination mit den geometrischen Verfahren erreichen.

- Eine Verbesserung der auf einen geringen Schnitt ausgerichteten Methoden läßt sich durch *Spektral-Methoden* erreichen, die in Form globaler Heuristiken einen Graphen in zwei Sektionen aufteilen und dabei die Probleme der Breitensuche vermeiden. Für die vorliegenden sehr großen Graphen sind die Methoden jedoch aufwendig. Da rein auf den Verbindungsschnitt ausgelegte Verfahren, wie nachfolgend gezeigt wird, keine guten Partitionierungen geliefert haben, wurden diese Ansätze nicht weiter betrachtet. Methoden, die in erster Linie den *Verbindungsschnitt optimieren*, sind für konventionelle neuronale Netze interessant, in denen keine Ungleichverteilungen der Aktivität auftreten. Solche Verfahren sind z.B. im Rahmen des SENROB-Projektes an den Universitäten Düsseldorf und Bonn [KSE92][Kre93][KSS93] und unter Berücksichtigung des Hardware-Graphen an der Universität Paderborn durchgeführt worden [Res99]. Im Zusammenhang mit Bildverarbeitungs-PCNN sind sie wenig hilfreich.

- Die als *Multi-Level-Techniken* zusammengefaßten Verfahren sind auch für sehr große Graphen geeignet. Verfahren aus diesem Bereich sind in der am Heinz Nixdorf Institut entstandenen Partitionierungsbibliothek PARTY zusammengefaßt [PD97][Pre00]. Die Methoden lassen sich mit geometrischen Vorpartitionierungen verbinden [Pre97-00], um so auch unter Berücksichtigung der Aktitvitätsverteilung zu Partitionen mit geringem Verbindungsschnitt zu gelangen.

Kapitel 5 - Architektur eines parallelen Simulatorsystems

Im vorangegangenen Kapitel wurde ein Satz von Methoden zur parallelen Simulation großer PCNN vorgestellt und in den Kontext des auf einer Verteilung der Neuronen basierenden Grundverfahrens gestellt. Es haben sich dabei drei Varianten des Verfahrens herauskristallisiert, die auf einer Kommunikation mit Hilfe der PreSpikes beruhen und sich im wesentlichen durch die Art der Topologiespeicherung unterscheiden. Daraus ergibt sich die folgende Einteilung:

- Das Verfahren kann mit einer *zentralen Topologiespeicherung* für beliebige Netzstrukturen (*non regular connections* - *nrc*) und einem *EIB-Cache* auf den Rechenknoten realisiert werden.
- Die Simulation kann mit einer *verteilten Topologiespeicherung* auf den Rechenknoten und einem PreSpike-Versand über Ziellisten erfolgen.
- Bei begrenztem lokalen Speicher auf den Rechenknoten kann dieses Verfahren auch auf Basis von *EIB-Masken* (*regular connections* - *rc*) durchgeführt werden.

Alle drei Verfahren arbeiten mit einer statischen Verteilung der Neuronen vor dem Simulationsstart und sind somit auf eine geeignete Netzpartitionierung angewiesen. Um die geschilderten Simulations- und Partitionierungsverfahren näher untersuchen zu können, sind diese im Rahmen der Arbeit programmtechnisch implementiert worden. Dabei sollten zum einen grundsätzliche Daten gewonnen werden, die eine Einschätzung der im Rahmen der Arbeit ebenfalls konzeptionierten Spezialhardware zulassen und Tendenzen bezüglich der prinzipiellen Eignung von Bildverarbeitungs-PCNN für eine parallele Simulation aufzeigen. Auf der anderen Seite ist so aber auch ein Satz von Werkzeugen für die parallele Simulation von PCNN auf Standardrechnern entstanden.

Die zu diesem Zweck zu lösenden Aufgaben sind im wesentlichen auf zwei Blöcke verteilt worden. Alle Aufgaben, die vor der eigentlichen Simulation lösbar sind, also vor allem die Partitionierung des neuronalen Netzes und die Erzeugung geeigneter Simulationsdaten, sind in Form eines *Netzcompilers* implementiert worden. Dadurch war es möglich, den eigentlichen *parallelen Softwaresimulator* sehr schlank zu halten.

5.1 Paralleler Simulator auf Basis von PVM

Zur Implementierung des Softwaresimulators wurde auf ein Werkzeug zurückgegriffen, das aus einer Gruppe von Standardrechnern in einem lokalen Netzwerk einen virtuellen Parallelrechner generiert. Dieses System heißt PVM (Parallele Virtuelle Maschine) und wurde im Rahmen eines Projektes an verschiedenen amerikanischen Universitäten entwickelt [GBD94]. Da im Labor des Fachgebietes Grundlagen der Elektrotechnik ein lokales Netzwerk mit bis zu zwölf etwa gleichartigen Sun-Workstations zur Verfügung stand, wurden auf Basis des PVM-Systems im Rahmen von zwei Diplomarbeiten [Cib99][Weg00] die beschriebenen Verfahren in Form von Softwaresimulatoren für diesen Workstation-Cluster implementiert. Da PVM neben den Sun-Workstations eine breite Palette anderer Hardwareplattformen einschließlich der meisten kommerziellen Parallelrechnersysteme unterstützt, ist der Einsatz der Simulatoren auf solchen Plattformen prinzipiell möglich.

Das PVM-System ist in zwei wesentliche Komponenten aufgeteilt (Abb. 5-1). Zum einen existiert ein Dämon-Programm, welches auf allen in den virtuellen Parallelrechner eingebundenen Rechenknoten gestartet wird. Diese *PVM-Dämonen* kommunizieren untereinander durch Botschaften, die in Puffern eingelagert werden. Mit Hilfe einer *Funktionsbibliothek* können zum anderen Benutzerprogramme mit den PVM-Dämonen

kommunizieren.

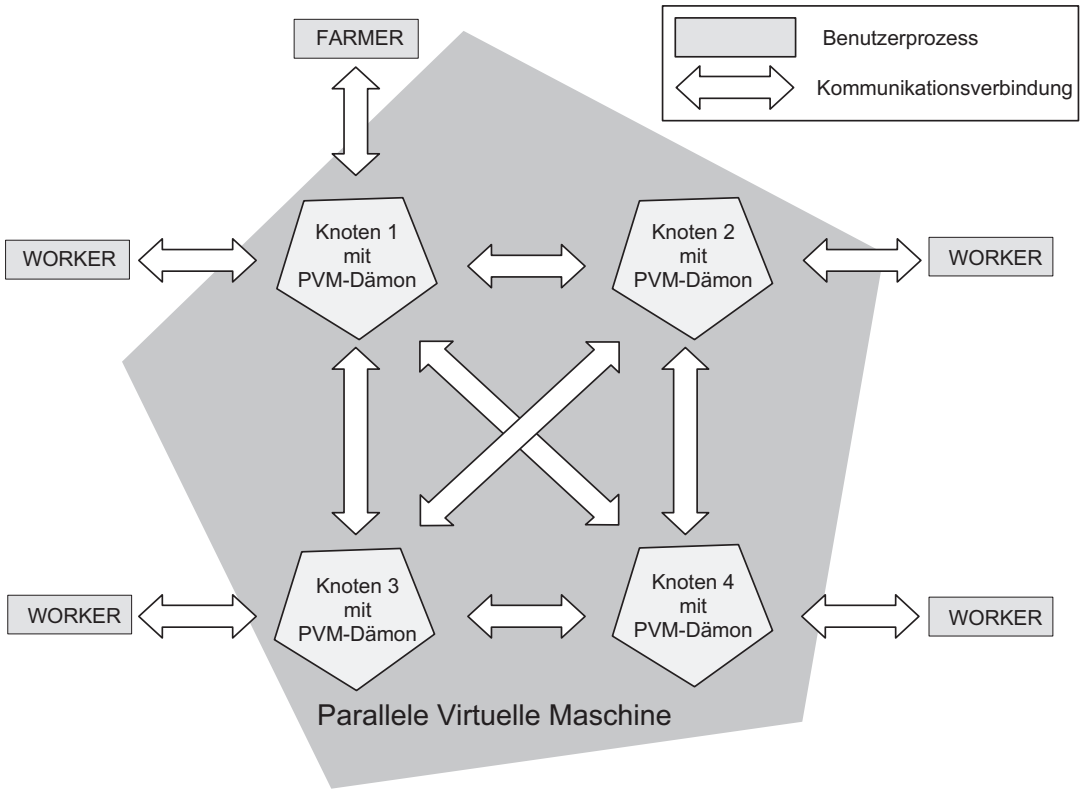

Abb. 5-1 Aufbau des PVM - Systems [Weg00]

Zur Implementierung des PCNN-Simulators wurde für die Benutzerprogramme eine sogenannte *Farmer-Worker-Konfiguration* gewählt [Cib99][Weg00]. Dabei stellt das Farmer-Programm die zentrale Synchronisationsinstanz dar, während die Worker-Programme die Rechenknoten des parallelen Verfahrens (siehe Abb. 4-1) bilden. Die Programme sind in der Programmiersprache C implementiert und greifen über Aufrufe aus der PVM-Funktionsbibliothek auf die Kommunikationspuffer der Dämonen zu. Jedem innerhalb von PVM verschickten Datenpaket ist die Kennung des Senders hinzugefügt. Zudem kann mit dem Paket eine Botschaft aus einer definierten Botschaftenmenge verbunden werden, d.h. leere Datenpakete können als reine Botschaften versendet werden. Die Größe der PVM-Puffer ist im wesentlichen durch den Speicher der Maschine bestimmt, auf der die Puffer abgelegt sind.

Implementiert wurden die drei geschilderten Verfahren (siehe oben), wobei sich die *rc-Variante* in der Realisierung kaum von der *Variante mit verteilter Topologiespeicherung* unterscheidet. In diesen beiden Simulatoren (siehe Abb. 4-6) werden Abkling- und Erregungsphase vollständig durch die Worker-Programme berechnet. Über das PVM-System werden PreSpike-Listen und Synchronisationsbotschaften verteilt. Die Aufgabe des Farmer-Programms besteht darin, durch ein Signal an alle Worker die Abklingphase anzustoßen und dann Spikelisten von den Workern zu sammeln und externe Spikes an die Worker

zu versenden. Haben die Worker die Abklingphase beendet und ihre Spikes verschickt, teilen sie dieses dem Farmer mittels einer Botschaft mit. Gleichzeitig berechnen die Worker nach Abschluß ihrer lokalen Abklingphase für die lokal erzeugten und die empfangenen Spikes über die lokal gespeicherten EIBs die Erregungsphase. Wurden alle Spikelisten ausgetauscht, so sind damit auch die Erregungen abgearbeitet, so daß der Farmer durch ein Signal an alle Worker die nächste Abklingphase und mithin den nächsten Zeitschritt einleitet.

In der Simulatorvariante mit *zentraler Topologiespeicherung* (*nrc-Variante*) sind die Aufgaben des Farmers umfangreicher (siehe Abb. 4-5 und Abb. 5-2). Wie bei den anderen Varianten leitet er durch eine Botschaft an alle Worker die Abklingphase ein. Für die zentrale Spikeliste empfängt der Farmer in der Abklingphase alle PreSpikes von den Workern. Zudem trägt er die externen Spikes dort ein. In der Erregungsphase liest der Farmer aus den zentralen Topologiedaten die EIBs zu den feuernden Neuronen und verteilt die dort enthaltenen Daten an die entsprechenden Worker. Ob der Worker, der den PreSpike erzeugt hat, die entsprechenden Daten in seinem lokalen EIB-Cache gefunden hat, oder ob er die Zusendung durch den Farmer erwartet, wird von ihm durch eine entsprechende Kennung im PreSpike mitgeteilt. Nach Empfang aller Spikelisten von den Workern und Abarbeitung der Erregungsphase leitet der Farmer wiederum durch Versand einer Botschaft an alle Worker die nächste Abklingphase und den nächsten Zeitschritt ein.

In den Simulatoren existieren damit zwei Synchronisationspunkte innerhalb eines Zeitschritts. Begonnen wird der Zeitschritt durch das zentrale Signal des Farmers. Dieser wartet dann auf die Nachricht der Worker, daß dort die Spikeerzeugung abgeschlossen ist. Hat er die Botschaft von allen Workern empfangen, so arbeitet er seine eigenen Aufgaben ab, um danach durch eine erneute Botschaft an alle Worker den Zeitschritt weiterzuschalten. Lokal wird der Zeitschritt von den Workern natürlich erst dann weitergeschaltet, wenn sie ihre Aufgaben aus dem alten Zeitschritt bearbeitet haben. Dieses ist möglich, da sie nicht mehr kommunizieren. Damit ist eine gewisse Aufweichung der starren Synchronisation gegeben, da

die Worker lokal zu unterschiedlichen Zeitpunkten in den nächsten Zeitschritt eintreten.

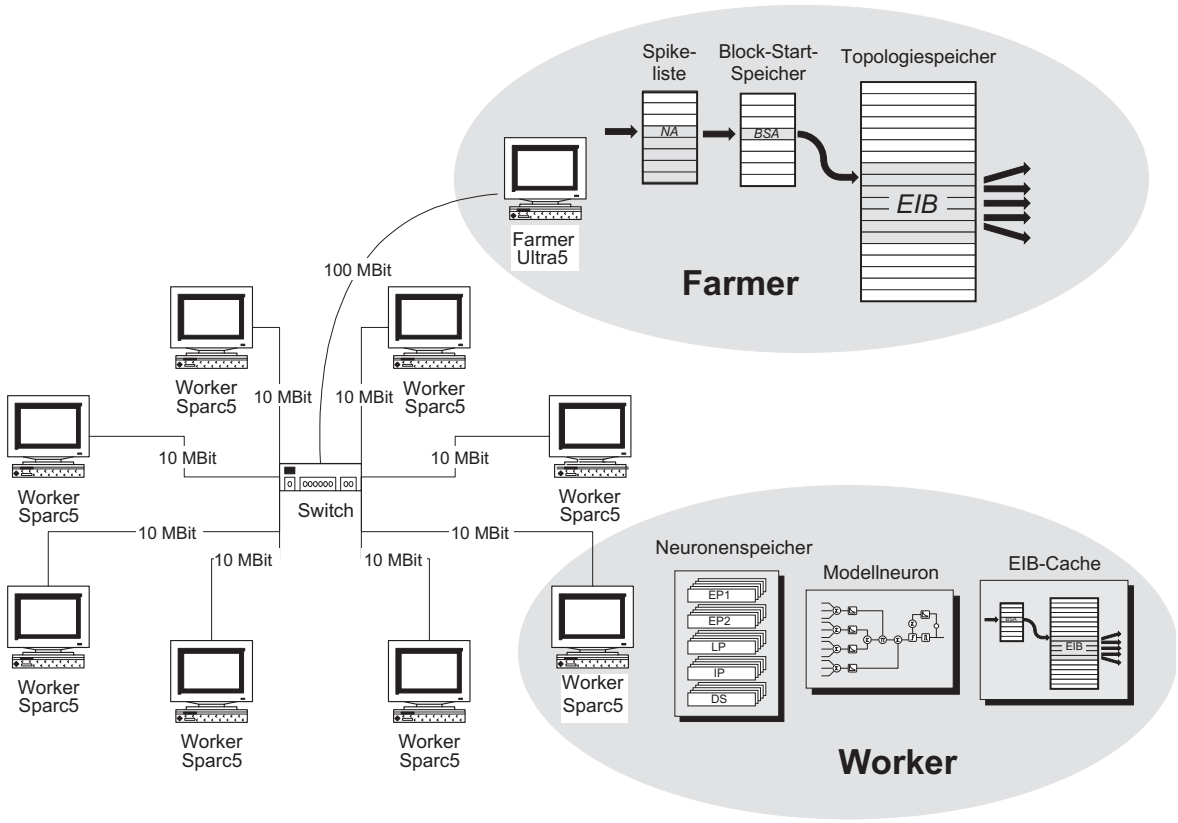

Abb. 5-2 PVM-Umgebung am Beispiel der nrc-Variante

Möglich wird dieses Vorgehen durch die Struktur des Farmer- und Worker-Programms. Zunächst einmal ist für jedes Objekt der Simulation, z.B. für ein Neuron, einen EIB, einen Abklingverlauf oder eine Zielliste, ein entsprechendes Datenobjekt erstellt worden. Auf dem Farmer bzw. auf dem Worker existiert dann ein Zeigerfeld, in dem unter der Neuronenadresse eines Neurons ein Zeiger auf dessen Datenobjekte gefunden werden kann. In der nrc-Variante des Simulators befinden sich dort auf dem Farmer Zeiger auf die EIBs und auf dem Worker Zeiger auf die Neuronendaten. Für im Cache befindliche EIBs existiert auch auf dem Worker der entsprechende Zeiger. Zu jedem Datenobjekt sind außerdem Funktionen vorhanden, welche die Daten zur Initialisierung von der Festplatte lesen, in PVM-Kommunikationspuffer einpacken, von dort entpacken und sie in die Zeigerfelder einhängen. Die Datenpakete werden zudem mit einer Botschaftskennung versehen. Zusätzlich zu den Zeigerfeldern sind Spike- und Abklinglisten vorhanden.

Die Botschaftskennungen können als Ereignisse aufgefaßt werden. Im Farmer- sowie im Worker-Programm ist daher eine Programmschleife implementiert, die auf ein Datenpaket bzw. eine Botschaft wartet und dann entsprechend der Botschaft eine Verarbeitungsprozedur anstößt. Wird z.B. zur Initialisierung einem Worker vom Farmer der Datenblock eines Neu-

rons zugesandt, so wird dort eine Prozedur angestoßen, die den Block in das Zeigerfeld einsortiert. Solche Botschaften heißen auch *Event-Messages*, so daß die Bearbeitungsschleife *Event-Handler* genannt wird. Nachdem ein Event bearbeitet ist, wird im Rahmen der Schleife auf das nächste Event gewartet. Wird während einer Event-Behandlung eine neue Botschaft empfangen, so wird diese chronologisch im PVM-Puffer des Dämons abgelegt. Damit kann der Worker zunächst die Bearbeitung des vorherigen Events beenden ohne das der sendende Farmer darauf warten muß. Diese Eigenschaft kann auch für eine weitere Aufweichung der Synchronisation genutzt werden und somit Ungleichverteilungen der Simulationslast entzerren.

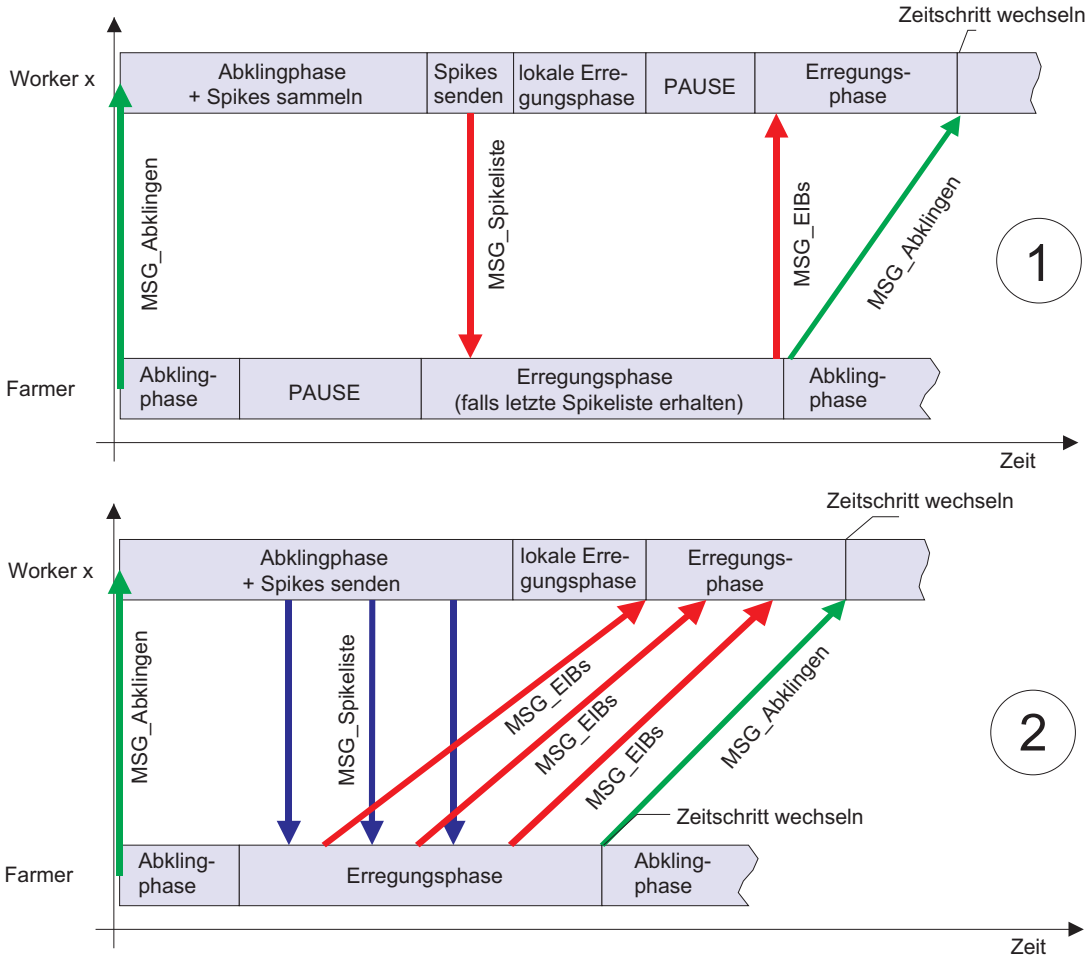

Abb. 5-3 Kommunikation und Synchronisation im PVM-Simulator

In Abb. 5-3 ist für die nrc-Variante des Simulators unter (1) zunächst zu sehen, wie der Farmer den Zeitschritt mit einer Botschaft *MSG_Abklingen* anstößt. Der Worker berechnet als Reaktion lokal die Abklingphase und sammelt die Spikes. Diese werden dann zusammen an den Farmer geschickt (*MSG_Spikeliste*), der daraufhin mit der Erregungsphase beginnt. Der Worker kann lokal die Erregungsphase für die EIBs aus dem Cache abarbeiten und wartet dann auf EIBs vom Farmer (*MSG_EIBs*). Nachdem der Farmer diese verschickt hat, zeigt er

dem Worker mit einem erneuten *MSG_Abklingen* den Zeitschrittwechsel an. Neben den möglichen Pausen bei diesem Vorgehen entstehen auch Schwerpunkte im Kommunikationsaufkommen. Durch eine Modifikation der Kommunikation (siehe Abb. 5-3 unter 2) können diese Schwerpunkte entschärft werden und Pausen lassen sich ggf. vermeiden. Dazu werden Spikes vom Worker zu kleineren Paketen gesammelt und während der gesamten Abklingphase zum Farmer verschickt. Dieser ermittelt sofort die EIBs und sendet diese zum Worker, wo sie im Empfangspuffer des PVM-Dämons bis zur Abholung gelagert werden. Der Worker kann dann direkt nach der lokalen Erregungsphase mit der Auswertung der EIBs beginnen. Indem während der gesamten Abklingphase Spikelisten verschickt werden, kann auch die Kommunikation in der rc-Variante bzw. der Variante mit verteilter Topologiespeicherung optimiert werden. Hier haben sich zudem wesentliche Vorteile dadurch ergeben, daß die PreSpikes nicht im Broadcast (siehe Kap. 4.2) versendet werden, sondern gezielt durch die Verwendung der Ziellisten auf den Workern. Insgesamt haben diese in [Weg00] vorgenommenen Optimierungen dazu geführt, daß die PVM-Simulation auf zunehmender Rechnerzahl gegenüber [Cib99] wesentlich besser skaliert.

Neben den Ergebnissen bezüglich der Lastverteilung und Kommunikationslast, die aus den PVM-Simulationen zur Einschätzung der Partitionierungsverfahren und des parallelen Ansatzes gewonnen und zur Leistungsabschätzung für das Spezialhardwarekonzept benutzt werden sollten, ist es natürlich auch interessant, inwieweit der erstellte Softwaresimulator geeignet ist, die Simulation von PCNN zu beschleunigen. Dazu wurden mit einer im weiteren erläuterten Partitionierung Messungen der Simulationsgeschwindigkeit auf unterschiedlichen Rechnerzahlen durchgeführt. Benutzt wurden Maschinen der Modellreihen Sparc 5 und Sparc 4 mit jeweils 64 MB Hauptspeicher, die in etwa vergleichbare Leistungsdaten aufweisen. Die Maschinen waren über 10 MBit Ethernet-Verbindungen an eine Sun Ultra 5 angebunden (siehe Abb. 5-2), die als Maschine für den Farmer diente. Diese Maschine wurde verwendet, da sie die Festplatte zur Ausgabe der Statistikdaten enthält und somit keine zusätzliche Netzlast durch Protokollausgaben erzeugt. Die Rechenlast des Farmers ist gegenüber den Workern marginal, d.h. ohne Statistikausgaben werden mit einer Sparc 5 als Farmer die gleichen Geschwindigkeiten erzielt. Leistungsdaten der Maschinen sind in Kap. 3.5 zu finden. Die Worker-Maschinen sind aufgrund ihres ausreichend großen Arbeitsspeichers in der Lage, die Variante mit verteilter Topologiespeicherung zu simulieren, weswegen auf die Messung der bezüglich der Kommunikation und Lastverteilung identischen rc-Variante verzichtet wurde. Das Weitzelnetz wurde auf bis zu acht Maschinen simuliert, für das Brausenetz wurden bis zu zwölf Maschinen eingesetzt. Die dabei genutzten Partitionsgrößen befinden sich in den für das spätere Hardwarekonzept relevanten Größenordnungen.

Die Messungen (siehe Abb. 5-4) zeigen, daß das Verfahren selbst auf einem Workstation-Cluster sehr gut skaliert. Unter Berücksichtigung der Tatsache, daß ein kommunikationsintensives Problem auf einer Plattform simuliert wird, die schon rein physikalisch mit weniger

als 1 MByte/s Daten austauscht und zudem aufgrund der aufwendigen Protokolle und Betriebssystemaufrufe für die sehr kleinteilige Kommunikation denkbar ungeeignet ist, lassen die Meßergebnisse erkennen, welches Potential in der parallelen PCNN-Simulation steckt. Für die Simulatorvariante mit gemeinsamer Netztopologie ergibt sich für das Weitzelnetz beim Wechsel von vier auf acht Maschinen immerhin ein Geschwindigkeitszuwachs um den Faktor 1,76. Ein paralleler Cluster von acht Sparc 5 Maschinen ist damit so schnell wie die gegenüber der Sparc 5 sechsmal schnellere Ultra 2 (siehe Kap. 3.5).

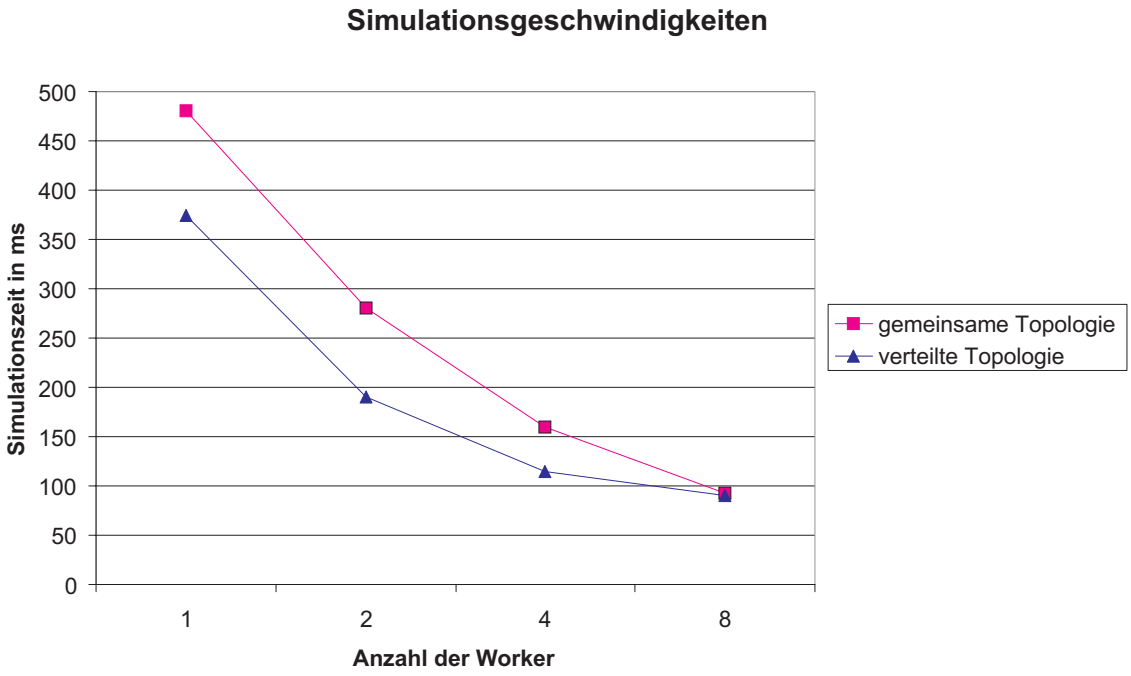

Abb. 5-4 Geschwindigkeitsmessungen mit dem Weitzelnetz (ms/Zeitschritt)

Damit ist der PVM-Softwaresimulator nicht nur für Untersuchungen zur parallelen PCNN-Simulation geeignet, sondern kann auch zur Senkung der Simulationszeiten bei der PCNN-Entwicklung benutzt werden. Das die Geschwindigkeitszuwächse auf einem dafür nicht ausgelegten Verbund von Workstations in einem lokalen Netz erreicht wurden, macht den Simulator auf der einen Seite universell einsetzbar, läßt aber auf der anderen Seite ein erhebliches Potential auf Parallelrechnern erwarten. Die dort üblicherweise bei der Neuronale-Netze-Simulation erreichten Geschwindigkeitszuwächse [Res93b][Dit94][Str94] werden im Fall der PCNN-Simulation schon auf einem Workstation-Cluster erreicht, was auf das günstigere Verhältnis von Kommunikationsaufwand zu Rechenaufwand durch die Spike-basierte Kommunikation zurückzuführen ist. Dabei haben die in [Weg00] vorgenommenen Kommunikationsoptimierungen dazu geführt, daß auch beim Übergang von vier auf acht Maschinen im Gegensatz zu [Cib99] noch ein Geschwindigkeitszuwachs zu verzeichnen ist. Die Simulation des für eine Parallelisierung aufgrund seiner Struktur prädestinierten Brausenetzes (siehe Abb. 5-5) ergab sogar beim Übergang von acht auf zwölf Maschinen noch

einen Geschwindigkeitszuwachs um den Faktor 1,45, so daß von einer in diesem Bereich fast linearen Beschleunigung ausgegangen werden kann.

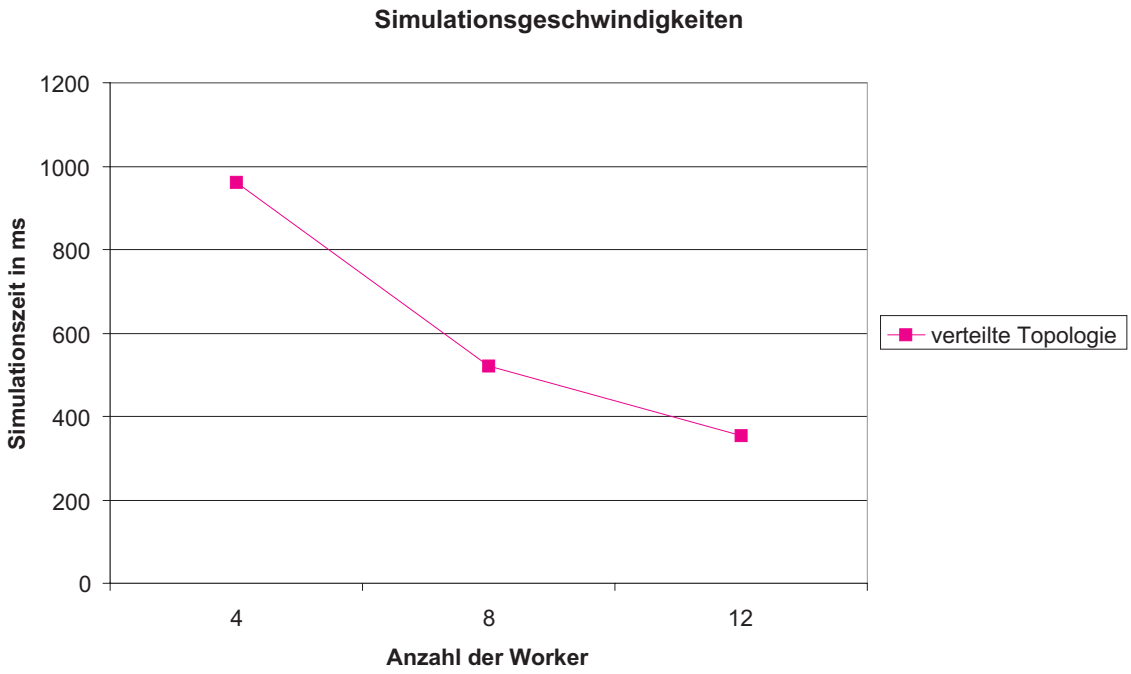

Abb. 5-5 Geschwindigkeitsmessungen mit dem Brausenetz (ms/Zeitschritt)

Insgesamt zeigen die Ergebnisse des PVM-Simulators, daß die parallele Simulation großer Bildverarbeitungs-PCNN vielversprechend ist und zumindestens in der Größenordnung von 20 bis 30 Rechenknoten bei einem parallelen Spezialrechner mit geeigneter Kommunikationshardware Geschwindigkeitszuwächse zu erwarten sind.

5.2 Netzbeschreibungssprache und Netzcompiler

Grundlage der guten Simulationsergebnisse ist eine Verteilung der Neuronen auf die Rechenknoten, die zu einer guten Lastverteilung und einer geringen Kommunikation führt. Eine solche Verteilung kann durch die in Kap. 4.4 beschriebenen Partitionierungsstrategien erreicht werden, wobei jedoch zum Teil genaue Kenntnisse über den Netzaufbau notwendig sind. Diese Kenntnisse können erlangt werden, wenn das Netz vom Entwickler in geeigneter Weise beschrieben wurde. Daneben soll die Art der Beschreibung natürlich auch dem Entwickler die Arbeit durch einen kompakten und strukturierten Aufbau erleichtern. Um diese Ziele zu erreichen, wurde im Rahmen des SPIKE128k-Projektes an der Universität Marburg die Neuronale-Netze-Sprache MNET entwickelt [Möl95][Möl96]. Für diese Sprache wurde außerdem ein Compiler implementiert, der aus einer MNET-Beschreibung die Initialisie-

rungsdaten für den SPIKE128k generiert. Der Versuch, den Compiler für die Zwecke der parallelen PCNN-Simulation anzupassen, mußte jedoch scheitern, da die Vorgehensweise des Marburger MNET-Compilers die Integration von Partitionierungsverfahren nicht zuließ. Dieser Compiler erzeugt nämlich im Laufe einer sequentiellen Abarbeitung der MNET-Beschreibung für jedes MNET-Konstrukt eine C-Funktion, die in ein großes C-Programm kopiert wird. Dieses Programm wird dann mit Hilfe eines C-Compilers zur Laufzeit des MNET-Compilers übersetzt und ausgeführt. Dabei werden dann die entsprechenden Daten für den SPIKE128k erzeugt. Damit liegt zu keinem Zeitpunkt eine komplette Beschreibung des neuronalen Netzes in den Datenstrukturen des Compilers vor, so daß auch keine Partitionierung angesetzt werden kann.

Aus diesem Grund wurde im Rahmen der vorliegenden Arbeit ein neuer MNET-Compiler konzipiert und u.a. im Rahmen von [Ibe00] implementiert. Dazu wurde zunächst die MNET-Sprache geringfügig modifiziert und neu spezifiziert [WSW98a], um die Überführung in eine kontextfreie Grammatik zu ermöglichen [Kas90]. Erst durch eine Sprachspezifikation in Form einer solchen kontextfreien Grammatik wird die Anwendung moderner Methoden des Compilerbaus ermöglicht. Diese Arbeiten wurden im Rahmen des Graduiertenkollegs „Parallele Rechnernetzwerke in der Produktionstechnik“ am Heinz Nixdorf Institut zusammen mit Kay A. Salzwedel durchgeführt [Sal98-00].

Mittels MNET wird ein neuronales Netz in zwei wesentlichen Schritten beschrieben. Zuerst werden alle benötigten Netzelemente definiert. Ausgehend von den Dendritenbäumen und Eingangsfiltern des Modellneurons (siehe Abb. 3-1) werden die benötigten Neuronentypen deklariert. Diese Neuronen werden dann zu zweidimensionalen rechteckigen Lagen angeordnet. Durch Kombination einer solchen Lage mit einer zweidimensionalen Auswahlfunktion (*map*) können nicht benötigte Neuronen ausgeblendet werden. Zur Verbindung zweier Lagen werden sogenannte *connection* Schemata definiert, bei denen es sich um mathematische Funktionen auf den Koordinaten der Quell- und der Zielneuronenlage handelt. Durch Einsetzen konkreter Koordinaten wird das Gewicht der Verbindung zwischen den Neuronen berechnet. Nachdem die einzelnen Netzelemente definiert sind, wird aus ihnen das Netz aufgebaut. Dazu werden die Neuronenlagen (*cluster*) im dreidimensionalen Koordinatensystem angeordnet. Unter Benutzung der *connection* Muster werden dann je zwei Lagen miteinander verbunden, wobei wiederum mittels der *map* Muster Neuronen der Quell- und Ziellagen von der Verbindung ausgenommen werden können. Der Sprachumfang wird ergänzt durch Konstrukte zur Festlegung der Abklingverläufe. Zudem sind den zentralen Konstrukten Ausdrücke zur Zuordnung aller benötigten Parameter angefügt. Der Sprachumfang ist in [WSW98a] dokumentiert, die verwendeten Netze liegen als Beispiele vor.

Der eigentliche MNET-Compiler ist als mehrstufiges modulares System aufgebaut und in Form mehrerer C-Funktionsbibliotheken implementiert. Durch Austausch oder Ergänzung

der Bibliotheken kann die Funktionalität modifiziert oder erweitert werden.

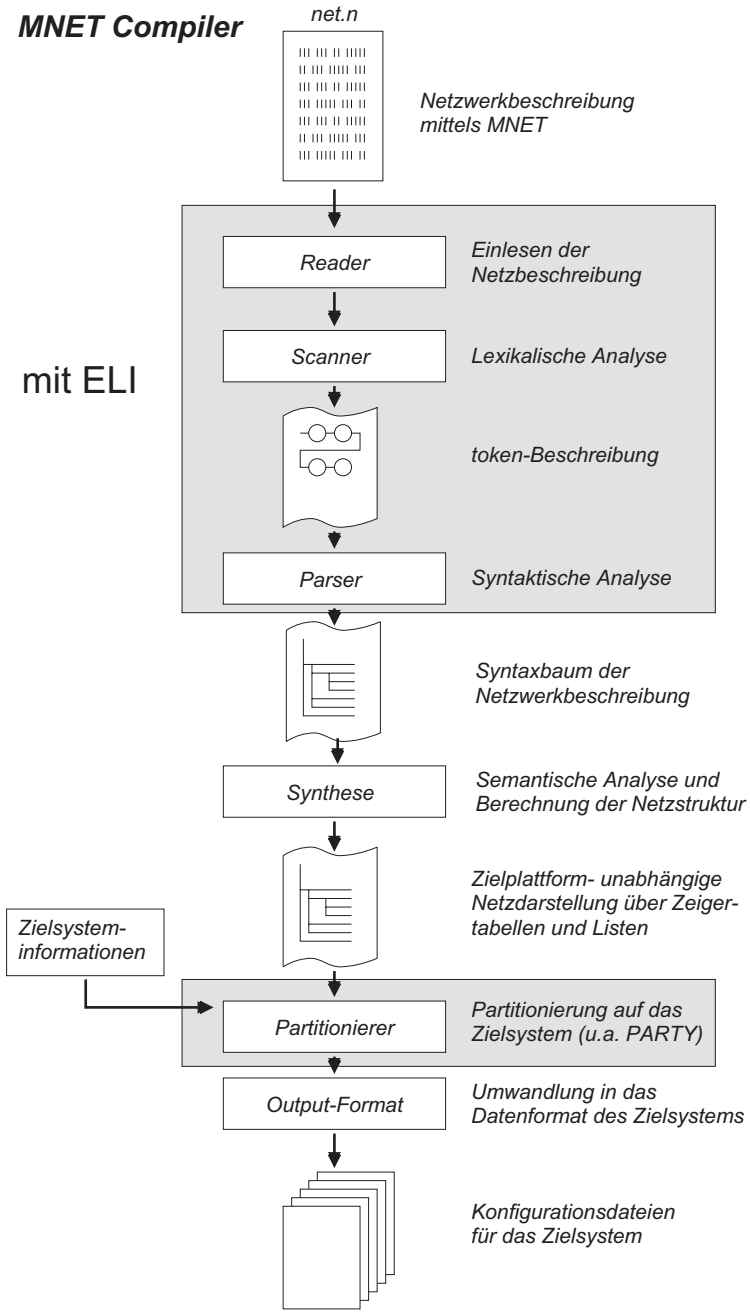

Abb. 5-6 Modularer Aufbau des Compilers für die MNET-Sprache

Zur Generierung einer Datenstruktur, die in Form eines Syntaxbaumes die Netzbeschreibung in analysierbarer Form wiedergibt, wurde ein Compiler-Frontend erstellt. Dazu wurde der Compiler-Compiler ELI (u.a. aus der Kastens-Gruppe des Fachbereichs Informatik der Universität Paderborn) verwendet [Kas90][GHL92], der aus einer kontextfreien Grammatik in Bakkus-Naur-Form (BNF) [Nau63] einen attributierbaren Syntaxbaum erzeugt, der dann

durch Attribute und Funktionen ergänzt den analysierbaren Syntaxbaum generiert. Zu detaillierteren Darstellungen sei auf [WSW98a][WSW98b][Ibe00] verwiesen.

Ein weiterer Syntheseblock des Compilers überführt die so erzeugte Darstellung der Struktur der MNET-Netzbeschreibung in eine Repräsentation des neuronalen Netzes, die Partitionierungen und die Erzeugung von Ausgabeformaten zuläßt. Ein wesentlicher Punkt bei dieser Überführung ist die Berechnung der *map* und *connection* Schemata aus der mathematischen MNET-Beschreibung. Zur Definition solcher Schemata ist annähernd der gesamte Umfang der mathematischen Funktionen in ANSI-C Notation unter Benutzung der math.h-Erweiterung erlaubt. Im MNET-Compiler ist ein Funktionsinterpreter integriert, der auf dem Syntaxbaum einer solchen ANSI-C Funktion Berechnungen ausführen kann. Die berechneten Funktionen werden in Form von Feldern oder zeigerverketteten Listen für die Verbindungen abgelegt. Im Grundsatz ist die erzeugte plattformunabhängige Netzdarstellung eine Struktur aus zeigerverketteten Listen, auf die zudem über Zeigertabellen zugegriffen werden kann, so daß Suchoperationen auf den Listen nicht notwendig sind. Ergänzt wird diese Struktur durch eine ähnlich aufgebaute Darstellung der Verknüpfungstopologie, die zudem in einer sender- und einer empfängerorientierten Form vorliegt. Aus dieser Netztopologie können die EIBs oder alternativ die rc-EIBs erzeugt werden.

Auf Basis der beschriebenen Datenstrukturen kommt dann der Partitionierer zum Einsatz, der in Form einer Funktionsbibliothek verschiedene Strategien zur Verteilung der Neuronen auf die Rechenknoten bereitstellt. Ergebnis der Partitionierung ist eine Umsetzung der Neuronennummern auf Neuronenadressen bzw. die Zuweisung dieser Adressen an Rechenknoten. Entsprechend werden dann ggf. die EIBs der Topologie aufgeteilt. Als Partitionierungsstrategien wurden in Anlehnung an die Einteilung in Kap. 4.4 die folgenden Verfahren untersucht.

1. Eine alternierende *Modulo-Verteilung* der einzelnen Neuronen und von Gruppen von je 256 Neuronen als Beispiel für *einfache Partitionierungsverfahren*.

2. Ein *Ausschneiden von quadratischen Gebieten* aus den Neuronenlagen und die Zuordnung von Gebieten gleicher rezeptiver Felder zu gleichen Rechenknoten. Diese Gruppen von Gebieten, bei denen die Gebietsgröße variiert werden konnte, wurden im Modulo-Verfahren auf die Knoten verteilt. Das Verfahren gehört zu den *geometrischen Verfahren* mit vertikalem Schnitt.

3. Eine *Modulo-Verteilung ganzer Neuronenlagen* bzw. -cluster. Da zusammenhängende Lagen verteilt werden, handelt es sich um ein *geometrisches Verfahren* mit horizontalem Schnitt.

4. Ein Verfahren, das bis zu einer bestimmten Tiefe alle *Nachfolgeneuronen* eines Neurons zu einer Partition zusammenfaßt. Dieses Verfahren ist den Methoden der *Breitensuche*

zuzuordnen.

5. Eine Kombination der Verfahren 2 und 4, um die gute Lastverteilung aus 2 mit dem guten Verbindungsschnitt aus 4 zu kombinieren. Dazu werden die Ausgangsneuronen der Breitensuche gemäß Verfahren 2 ausgewählt.

Die weiteren Verfahren benutzten eine spezielle Bibliothek zur Graphenpartitionierung - die PARTY-Bibliothek [PD97]. Diese Bibliothek ist z.T. im Rahmen des Graduiertenkollegs von Robert Preis erstellt worden [Pre97-00]. Einzelne Funktionen wurden von ihm speziell für die Anforderungen der vorliegenden Arbeit angepaßt. Die Methoden der PARTY-Bibliothek sind den *Multilevel-Verfahren* zuzuordnen. Dabei wird zuerst das als Graph dargestellte Netz in mehreren Stufen auf einen Graphen mit geringerem Knotengrad reduziert. Es werden jeweils zwei zusammenhängende Knoten zu einem neuen Knoten zusammengefaßt. Dieser Vorgang wird wiederholt, bis die Knotenanzahl in einer gewünschten Größenordnung liegt. Auf diesem kleinen Graphen kann dann eine optimale Partitionierung schnell gefunden werden. Danach wird der Graph wieder in mehreren Schritten expandiert, wobei in jedem Schritt die Partitionierung mit lokalen Methoden [MDP96] optimiert wird. Zu näheren Angaben sei auf die Dissertation [Pre00] verwiesen. Verwendet wurden die Funktionen in zwei Verfahren.

6. Die Lagen des Netzes (cluster) wurden mit Hilfe der PARTY-Funktionen verteilt, so daß eine Kombination aus *geometrischen* und *Multilevel-Verfahren* entstand.

7. Die einzelnen Neuronen wurden mit Hilfe von Methode 2 vorpartitioniert und mit den PARTY-Funktionen verteilt.

Aufgrund der fast beliebig großen Anzahl von Möglichkeiten der Kombination und Variation der Verfahren ist es im Rahmen der vorliegenden Arbeit nicht möglich, die Partitionierungsproblematik erschöpfend zu behandeln. Es werden im folgenden vielmehr nur einige Beispiele gegeben, die Tendenzen aufzeigen und Werte für weitere Abschätzungen liefern.

5.3 Simulation von Beispielnetzen

Im weiteren wird zunächst auf den Compilierungsprozeß für die in Kap. 2.4 beschriebenen Beispielnetze eingegangen. Anschließend werden die protokollierten Ergebnisse der Simulation für einige Beispiele dargestellt. Da sich das Stoeckernetz bezüglich der parallelen Simulation aufgrund der geringen Netzgröße als wenig ergiebig gezeigt hat, wird auf eine Darstellung verzichtet. Abschließend werden die Ergebnisse gewertet und eingeordnet.

5.3.1 MNET-Beschreibungen und Compilerläufe

Zunächst einmal ist von Interesse, wieviel Zeit der Compiler benötigt, um eine MNET-Beschreibung in simulierbare Daten zu transformieren, und welche Ressourcen dazu benötigt werden (siehe Tabelle 5-1). Die MNET-Beschreibung selbst umfaßt dabei für keines der Netze mehr als etwa 10 Seiten. Die Ergebnisse wurden auf einer Sun Ultra 5 (vergl. Kap. 3.5) mit 320 MB Hauptspeicher und ca. 600 MB Swap-Bereich ermittelt. Die Daten können abhängig von der Anzahl der erzeugten Partitionen leicht differieren.

compiliertes Netz	Hauptspeicherbedarf	Compilerlaufzeit	Ausgabedaten
Stoeckernetz	150 MB	7 min	30 MB
Weitzelnetz	150 MB	40 min	26 MB
Brausenetz	620 MB	20 h	114 MB

Tab. 5-1 Ergebnisse des Compilierungsprozesses

Insbesondere die Ergebnisse der Laufzeitmessung sind recht ernüchternd. Dabei ist die extrem lange Laufzeit der Compilierung des Brausenetzes auf den nicht ausreichenden Hauptspeicher der Sun Ultra 5 zurückzuführen. Die Maschine benötigt den überwiegenden Teil der Zeit zur Auslagerung von Daten in den Swap-Bereich. Zusätzlich zu den Laufzeitergebnissen sind in der Tabelle auch die Größen der erzeugten Simulationsdateien aufgeführt.

Bei einer Analyse der Verteilung der Simulationszeit und des Speicherbedarfs auf die einzelnen Schritte des Compilers zeigen sich zwei Schwerpunkte. Ein erheblicher Zeitbedarf entsteht bei der Berechnung der Verknüpfungs- und Auswahlschemata aus den map- und connection-Beschreibungen. Für jede dieser Beschreibungen wird eine Maske erzeugt, die dann für alle Neuronen einer Quellneuronenlage benutzt wird. Der Syntaxbaum der mathematischen Funktionsbeschreibung muß dazu vom Funktionsinterpreter für jede mögliche Zielkoordinate einmal durchlaufen werden, so daß erheblicher Aufwand entsteht. Die Anwendung der berechneten Maske auf alle Quellneuronen der Lage ist im Gegensatz dazu sehr schnell. Im Fall des Weitzelnetz wird für die Erzeugung der Masken etwa ein Drittel der Compilierungszeit benötigt. Dieser Aufwand kann drastisch reduziert werden, wenn die Größe einer Verbindungsmaske durch ein MNET-Sprachkonstrukt vorgegeben und so der Suchraum eingeschränkt wird. Weitere Verbesserungen können erreicht werden, indem der Compiler inkrementell nur geänderte Beschreibungsteile neu compiliert. Prinzipiell können die Schemata zudem auch parallel berechnet werden. Bis zur Erzeugung der plattformunabhängigen Netzdarstellung (siehe Abb. 5-6) werden insgesamt etwa 40% der Compilierungszeit benötigt. Der Hauptspeicherbedarf bewegt sich unterhalb von 20 MByte. Erst die Erzeugung der Netztopologie läßt den Bedarf derartig wachsen, wobei insbesondere die Berechnung der empfängerorientierten Topologiedarstellung erheblichen Aufwand verursacht. Diese Darstellung wird für das ParSPIKE-System und für die auf PARTY basierenden Par-

titionierungen benötigt. Dieser Teil des Compilers weist sicherlich das höchste Optimierungspotential auf. Die Partitionierungsverfahren haben dagegen einen Zeitbedarf von weniger als einer Minute. Die Module zur Ausgabe der Daten im Zielformat sind in ihrem Aufwand hauptsächlich durch die Geschwindigkeit des Plattenzugriffs bestimmt. Insgesamt erreicht der Compiler bei weitem nicht den Optimierungsgrad des PVM-Simulators, so daß sich neben der Ergänzung weiterer Partitionierungsverfahren Potential für umfangreiche weitere Arbeiten ergibt.

In Tabelle 5-2 ist die Ersparnis an Speicheraufwand aufgeführt, die durch die Darstellung der Vernetzungsstruktur mittels EIB-Masken (rc) erreicht werden kann.

Netz	Anzahl der Masken	Verbindungen insgesamt	größte Maske	kleinste Maske
Stoeckernetz	17	9.209	4.096	1
Weitzelnetz	88	33.384	16.384	1
Brausenetz	12	378	34	26

Tab. 5-2 Erzeugung von rc-EIBs durch den Compiler

Wird berücksichtigt, daß das Stoeckernetz bei vollständiger Topologiedatenerzeugung über 6.466.793 Verbindungen verfügt, das Weitzelnetz über 3.764.120 Verbindungen und das Brausenetz über 20.873.985 Verbindungen, so zeigt sich eine drastische Reduktion des Speicheraufwandes durch die Verwendung von rc-EIBs. Beim Stoeckernetz und beim Weitzelnetz resultieren die jeweils größten Masken aus der Implementierung einer globalen Inhibition (siehe Kap. 2.4), die ggf. auch algorithmisch gelöst werden kann, so daß diese Masken dann entfallen.

5.3.2 Ergebnisse der Partitionierungen

Die Bewertung der Partitionierungsverfahren erfolgt anhand von Meßwerten für die Lastbalancierung und den Kommunikationsaufwand. Für die Lastbalancierung wird der Füllstand der Abklingliste *a* und die Anzahl der Erregungen *e* herangezogen, da sie entsprechend Kap. 3.5 den Hauptaufwand der Simulation ausmachen. Die Kommunikation ist umgekehrt proportional zur Anzahl der Cache-Treffer h_1 im EIB-Cache bzw. zu den übertragenen Spikemengen h_2. In den abgebildeten Grafiken wird zu diesen Werten jeweils ein Mittelwert über alle Worker sowie ein Maximal- und ein Minimalwert als eine Art von Bandbreite angegeben. Dabei ist insbesondere der Mittelwert des am stärksten ausgelasteten Workers für die erreichbare Gesamtgeschwindigkeit relevant. Ermittelt wurden die Werte im wesentlichen für das Weitzelnetz. Das Stoeckernetz zeigt die gleichen Tendenzen wie das Weitzelnetz und ist zudem aufgrund seiner geringen Größe für das Spezialhardwarekonzept irrelevant. Das Brausenetz zeigt aufgrund seiner einfachen Struktur schon bei den einfachen Partitionie-

rungsverfahren sehr gute Ergebnisse, so daß es nur für zwei Verfahren betrachtet wird. Weitere Abbildungen zu den Verfahren befinden sich im Anhang B.

Als erstes Verfahren soll das im vorangegangenen Kap. 5.2 unter (3) beschriebene geometrische Verfahren mit horizontalem Schnitt in Form einer alternierenden Verteilung der Neuronenlagen auf die Rechenknoten vorgestellt werden [Ibe00][Weg00].

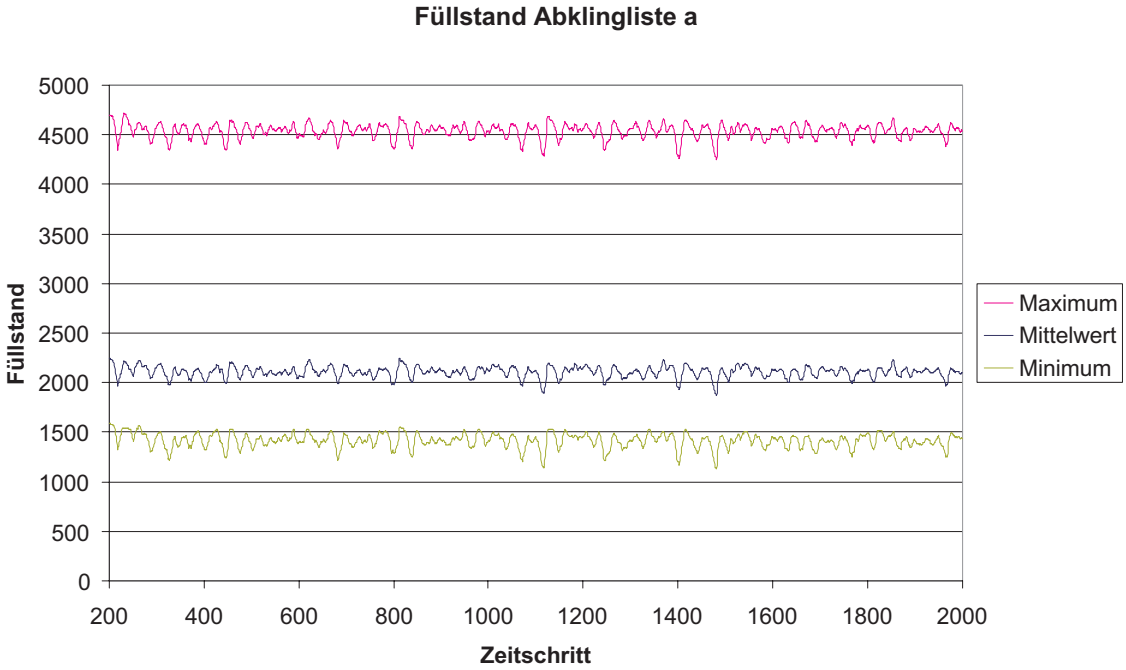

Abb. 5-7 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei lagenweiser Verteilung (3)

Dieses Verfahren erreicht schon in der Lastbalancierung schlechte Werte (Abb. 5-7). Der Füllstand der Abklingliste *a* ist für den meistbeschäftigten Worker mehr als doppelt so hoch wie der Mittelwert. Für die Erregungsphase und die Kommunikation ergeben sich ähnlich unbalancierte Werte, die Cachetrefferrate liegt im Mittel bei nur 13,7%. Das Verhalten ist darauf zurückzuführen, daß in den Lagen aufgrund der Reizspezifität unterschiedlich hohe Anzahlen von Neuronen aktiv sind. Diese Aktivitätsschwerpunkte werden durch das Verfahren nicht aufgebrochen sondern auf einen einzigen Worker partitioniert. In Untersuchungen der Eckhorn-Gruppe in Marburg [MSP97] ist gerade dieses Verfahren für die Verteilung von PCNN empfohlen worden. Allerdings wurde zum Test ein speziell erzeugtes Netz ohne konkrete Funktion benutzt. Mit einem realen Netz konnte auf einem IBM SP2 Parallelrechner mit 8 Prozessoren nur eine Geschwindigkeitssteigerung um den Faktor 1,25 erreicht werden. Bei dem synthetischen Netz lag dieser Wert für die lagenbasierte Verteilung bei 4,1.

Um die Bildung von Aktivitätsschwerpunkten konsequent aufzubrechen, sind unter (1) beschriebene einfache Methoden in Form einer alternierenden Verteilung der Neuronen auf die

Rechenknoten benutzt worden [Cib99], die auch in [JRS98] vorgeschlagen wird. Dabei zeigte sich eine fast ideale Lastverteilung mit einer Abweichung von +/- 5% bei 8 Workern. Da das Verfahren zu einem sehr großen Verbindungsschnitt führte, ergaben sich allerdings mit einer Cachetrefferrate von 13,2% schlechte Kommunikationswerte. Auch bei der Verteilung von Gruppen von 256 entsprechend der Numerierung aufeinanderfolgenden Neuronen konnte nur eine leicht verbesserte Cachetrefferrate von 14% erreicht werden.

Signifikant verbessert wurden diese Werte, wenn die alternierend verteilten Neuronengruppen nach einem geometrischen Verfahren ausgewählt wurden. Bei einer Zerteilung der Neuronenlagen in quadratische Ausschnitte der Größe 4x4 und einer Zuordnung der Ausschnitte mit gleichem rezeptiven Feld aus allen Lagen auf die gleichen Rechenknoten ergaben sich gute Lastverteilungswerte und ein zumindestens akzeptabler Verbindungsschnitt. Das Verfahren gehört zu den geometrischen Verfahren mit vertikalem Schnitt (2) und erzeugt Säulenschnitte aus den hierarchisch gestapelten Neuronenlagen. Durch die recht kleine Grundfläche der Säulen werden Aktivitätsschwerpunkte aufgebrochen, jedoch insbesondere die lokal begrenzten Verbindungsmuster zu Nachfolgelagen selten geschnitten.

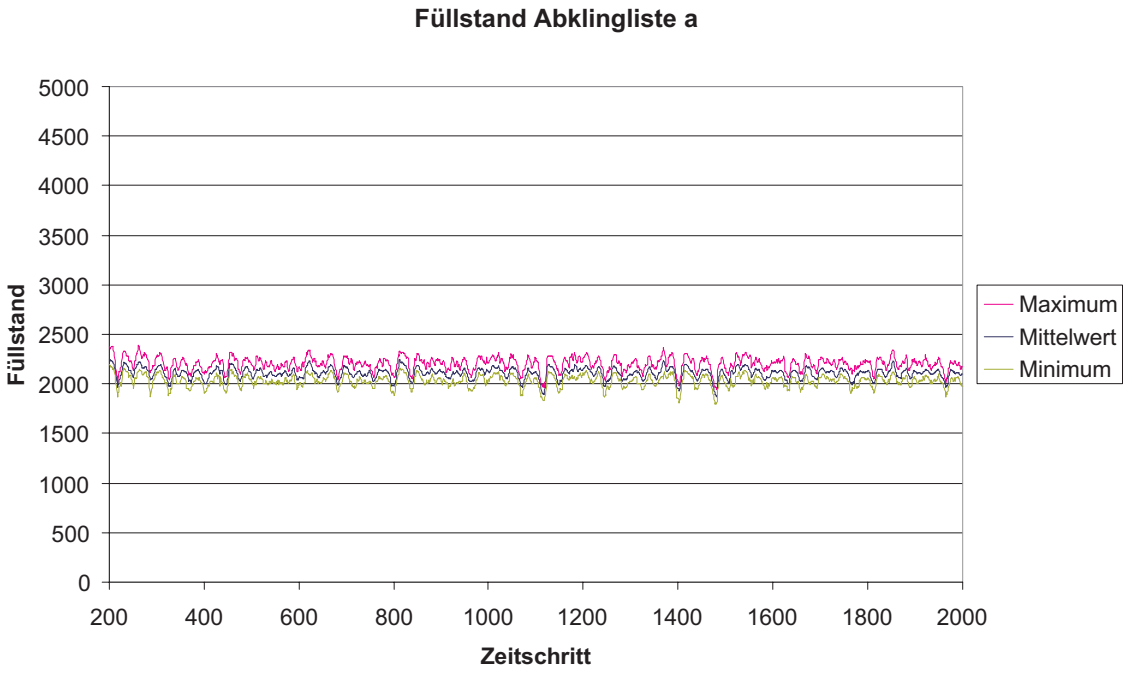

Abb. 5-8 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei säulenweiser Verteilung (2)

In der Abklingphase (Abb. 5-8) bewegt sich die Lastbalancierung wie bei der Säulen-Verteilung der Neuronen im Bereich von +/- 5%. Im Mittel liegt der Füllstand der Abklingliste *a* bei 2.107 Neuronen, der maximal ausgelastete Rechenknoten bearbeitet im Mittel 2.204 Neuronen. Eine ähnlich gute Lastbalancierung ergibt sich für die Auslastung in der Erre-

gungsphase, die durch die Anzahl der Erregungen *e* repräsentiert wird (siehe Abb. 5-9).

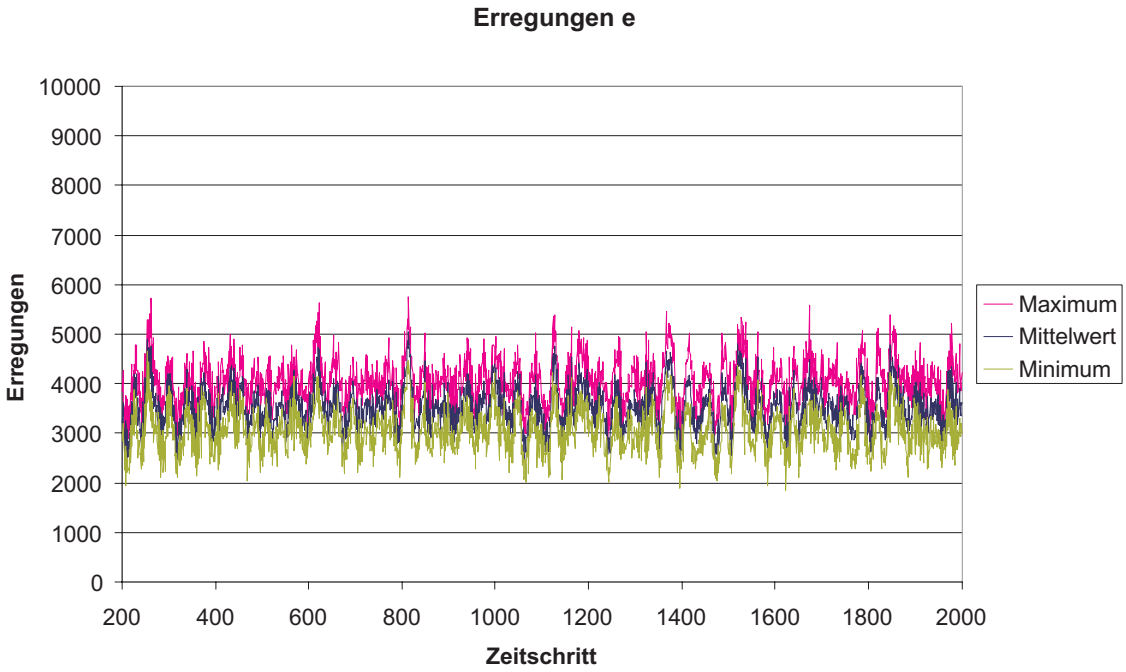

Abb. 5-9 Lastverteilung in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei säulenweiser Verteilung (2)

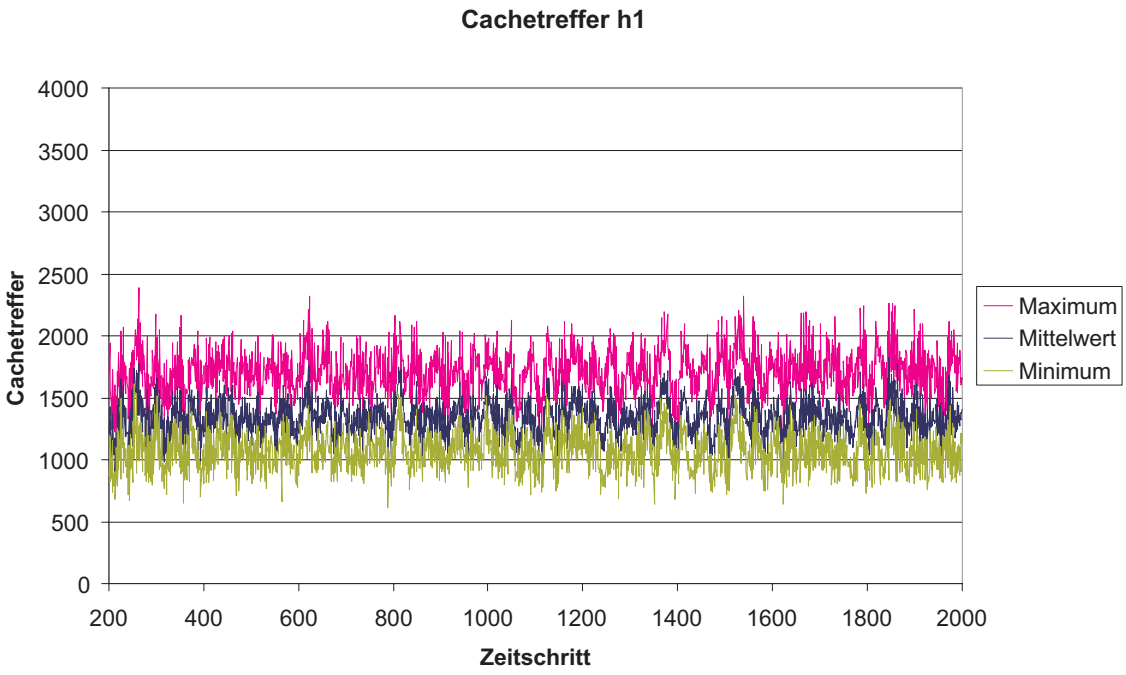

Abb. 5-10 Cachetreffer in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei säulenweiser Verteilung (2)

Im Mittel werden pro Rechenknoten 3.516 Neuronen erregt, die maximale Auslastung eines Knoten beträgt im Mittel 4.047 zu bearbeitende Erregungen *e*, d.h. sie liegt um 15% über dem Mittelwert. Von diesen Erregungen werden im Mittel 1.335 aus dem Cache bedient (h_1), was einer Trefferquote von 38% entspricht (Abb. 5-10). Diese Partitionierung des Weitzelnetzes ist für die Geschwindigkeitsmessungen in Kap. 5.1 verwendet worden (Abb. 5-4). Für weitere Abschätzungen werden die Maximalwerte der Auslastung für die Abklingphase a = 2.204 und für die Erregungsphase e = 4.047 benötigt. Weiterhin relevant sind die mittlere lokal verarbeitete Spikeanzahl h_2 = 50 und die mittlere kommunizierte Spikeanzahl s_{com} = 152.

Insbesondere aufgrund der nicht sonderlich guten Cachetrefferrate von 38% wurden in [Cib99] Untersuchungen mit einem Verfahren der Breitensuche (4) durchgeführt. Dabei wurden Nachfolgeneuronen bis zur Tiefe 16 zu Gruppen zusammengefaßt. Mit diesem Verfahren ließ sich bei 8 Maschinen die Cachetrefferrate für das Weitzelnetz im Mittel auf 59% steigern. Maximal konnten 77,8% der Erregungen aus dem Cache bedient werden. Leider ging diese Kommunikationsreduktion mit einer Verschlechterung der Lastbalancierung einher. Die Auslastung des meistbeschäftigten Rechenknotens lag in der Abklingphase um 59% über dem Mittelwert, in der Erregungsphase sogar um 114%. Damit erreichte das Verfahren in der Geschwindigkeitsmessung keine signifikanten Zuwächse. Zu beachten ist weiterhin, daß eine Breitensuche nicht grundsätzlich zu einem guten Verbindungsschnitt führt [Cib99], wie das Beispiel in der folgenden Abb. 5-11 zeigt.

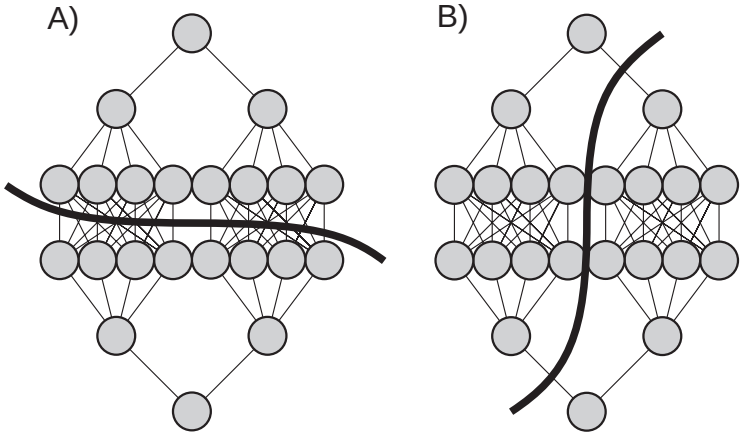

Abb. 5-11 Schlechter Verbindungsschnitt bei der Breitensuche (A) gegenüber gutem Schnitt bei der geometrischen Säulenmethode (B)

Die Kombination von Breitensuche und geometrischem Säulenverfahren (5) verbessert die Lastverteilung auf +/- 19% in der Abklingphase und +/- 43% in der Erregungsphase. Gleichzeitig sinkt jedoch die mittlere Cachetrefferrate auf 41%, so daß insgesamt eine Verschlechterung gegenüber der Säulen-Partitionierung entsteht. Ein ähnlicher Effekt wird erzielt, wenn statt der Säulen mit der Grundfläche 4x4 solche mit einer Grundfläche von 8x8 gewählt werden. Die mittlere Cachetrefferrate steigt dann auf 63% und damit auf den höchsten bei 8

Maschinen gemessenen Wert, die Lastungleichgewichte der am stärksten belasteten Knoten erreichen jedoch 22% des Mittelwertes für die Abklingphase und 50% für die Erregungsphase. Da die Lastverteilung für eine parallele Simulation in jedem Fall kritisch ist, während der Kommunkationsaufwand ggf. durch spezielle Verbindungsnetze abgefangen werden kann, ist der säulenbasierten geometrischen Verteilung der Vorzug zu geben.

Während die säulenbasierte Methode den Verbindungsschnitt nur implizit berücksichtigt (ein wesentlicher Teil der Verbindungen verläuft innerhalb der Säule), ist der Verbindungsschnitt bei den graphenbasierten Methoden (6) und (7) gerade das hauptsächliche Kostenkriterium bei der Erstellung von Verteilungen. Da jedoch aufgrund der Untersuchungen bekannt ist, daß dieses Kriterium in der Relevanz der Lastverteilung nachgeordnet ist, sind Methoden untersucht worden, die typische Graphenpartitionierungen mit geometrischen Vorpartitionierungen verbinden. Zu den Graphenpartitionierungen, die aus der Gruppe der Multilevel-Techniken gewählt wurden, sei auf [PD97][Pre00] verwiesen.

Die einfachste Kombination der Graphenpartitionierung mit den geometrischen Methoden ist die Verteilung ganzer Neuronenlagen unter Berücksichtigung der Verknüpfungsstruktur (6). Dabei konnten mit einer Cachetrefferrate von 22% gegenüber Methode (3) leichte Verbesserungen erreicht werden. Die maximalen Ausschläge der Lastverteilung lagen mit 95% in der Abklingphase und 150% in der Erregungsphase jedoch nicht in einem akzeptablen Bereich. Insgesamt erscheint eine lagenbasierte Verteilung für das Weitzelnetz ungeeignet.

Eine weitere Möglichkeit der Kombination wird durch eine Vorpartitionierung nach der säulenbasierten Methode (2) und eine nachfolgende Graphenpartitionierung der Neuronen gebildet. Diese Methode entspricht der Methode (4) der Breitensuche, wobei die Graphenpartitionierung die in Abb. 5-11 gezeigten Nachteile vermeidet. Insgesamt wurden jedoch auch hier mit einer Cachetrefferrate von 27% sowie einem Lastungleichgewicht von 13% für die Abklingphase und 26% für die Erregungsphase keine befriedigenden Werte erreicht. Verbesserungen können in Zukunft möglicherweise durch die Ergänzung der Kostenfunktion der Graphenpartitionierung um einen geometrischen Term erreicht werden.

Neben dem Weitzelnetz, daß sich durch aufwendige Verknüpfungsstrukturen zwischen Neuronenlagen und innerhalb dieser Lagen auszeichnet und zudem aufgrund der sehr stark spezialisierten Reizselektivität große Schwankungen der Netzaktivität aufweist, wurde auch das etwa dreimal größere Brausenetz partitioniert. Dieses Netz bildet eine Vorverarbeitungsstufe, die prinzipiell auch mit dem Weitzelnetz als hierarchisch nächster Stufe kombiniert werden könnte. Das Netz besteht aus 24 gleichartigen Lagen, die alle von zwei Eingangslagen gespeist werden und untereinander nicht verknüpft sind. Die Anzahl der Verknüpfungen zwischen den Eingangslagen und den eigentlichen Netzlagen ist für alle Lagen etwa identisch. Damit ist das Netz ideal für eine Parallelisierung geeignet. Tatsächlich erreicht schon das denkbar einfachste Verteilungsverfahren, nämlich die lagenweise Verteilung des Netzes

(3) im Gegensatz zum Weitzelnetz sehr gute Ergebnisse. Die Lastverteilung in der Abklingphase (Abb. 5-12) zeigt mit + 1,4% bzw. - 3,9% fast ideale Werte, der Mittelwert für *a* liegt bei 22.223, der Maximalwert bei 22.528.

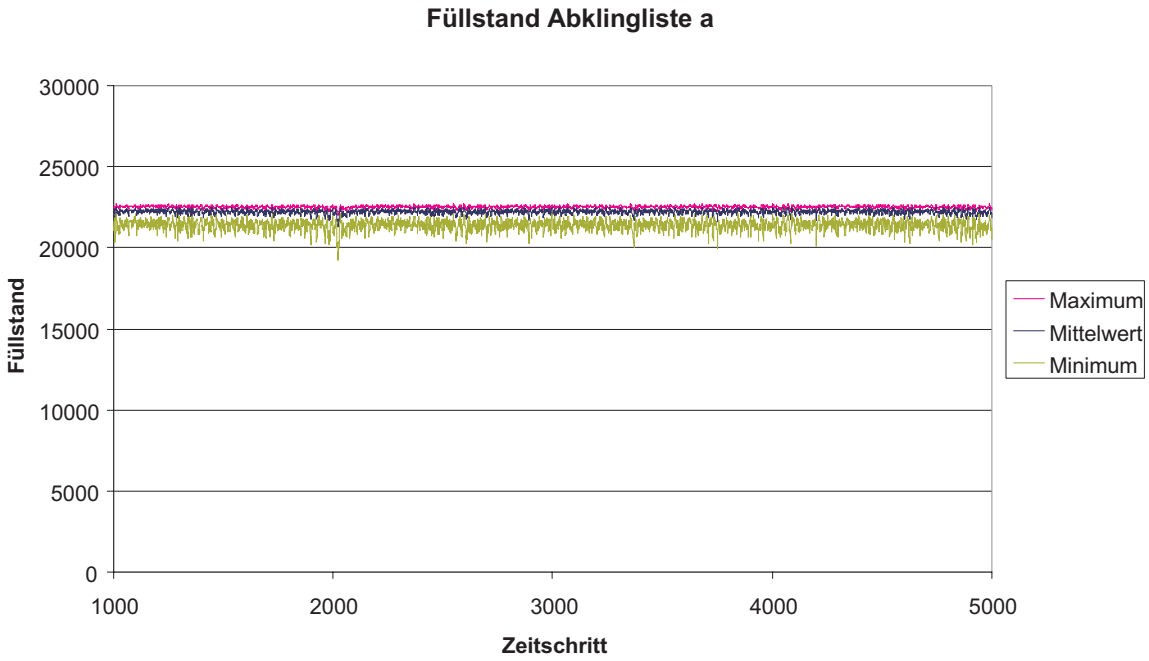

Abb. 5-12 Lastverteilung in der Abklingphase für die Simulation des Brausenetzes auf 12 Maschinen bei lagenweiser Verteilung (3)

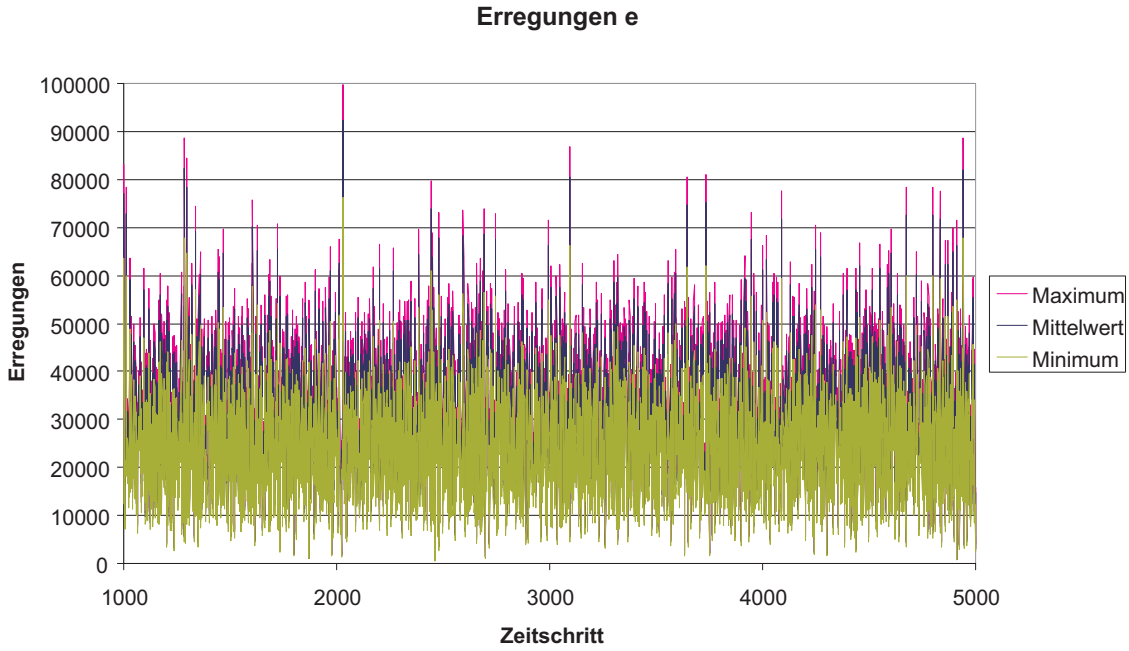

Abb. 5-13 Lastverteilung in der Erregungsphase für die Simulation des Brausenetzes auf 12 Maschinen bei lagenweiser Verteilung (3)

Die Verteilung in der Erregungsphase (Abb. 5-13) ist mit + 8% und - 21% schlechter, allerdings ist für die Rechenzeit des parallelen Systems der Maximalwert von *e* verantwortlich, der mit 30.901 nicht wesentlich über dem Mittelwert von 28.629 liegt. Dabei ist zu beachten, daß die Erregungsphase weniger Aufwand als die Abklingphase erzeugt.

Eine Überprüfung mit Hilfe der säulenbasierten Partitionierung (2), die für das Weitzelnetz gewählt wurde, brachte für das Brausenetz die gleichen guten Resultate, was angesichts der Ergebnisse der lagenweisen Methode, die auch als Säulenmethode mit maximaler Säulengrundfläche angesehen werden kann, wenig verwunderlich ist. Mit Hilfe der dargestellten Lagenmethode wurden auch die in Abb. 5-5 dargestellten Geschwindigkeitsmessungen durchgeführt.

5.3.3 Auswertungen und Folgerungen

Die Entwicklung und Überprüfung des parallelisierenden MNET-Compilers und des PVM-Softwaresimulators zeigen, daß die parallele Simulation großer PCNN sehr wohl Sinn macht. Insbesondere, wenn bedacht wird, daß die Resultate der Geschwindigkeitsmessungen auf einem dafür eigentlich wenig geeigneten lokalen Labornetz erzielt wurden und gerade nicht auf Hochleistungsparallelrechnern, und dann trotzdem die dort erzielten Ergebnisse [Dit94][MSP97][Res93b][Str94] übertroffen werden, so zeigt sich, daß das gewählte PCNN-Simulationsverfahren erhebliches Potential zur parallelen Berechnung aufweist. Die Exploration dieses Potentials auf kommerziellen Parallelrechnern bietet dabei sicherlich Möglichkeiten für diverse Folgearbeiten.

Dabei stellt der prototypisch erstellte MNET-Compiler [WSW98a][WSW98b][Ibe00] den wichtigsten Ansatzpunkt dar. Neben Optimierungen bezüglich des Ressourcenbedarfs des eigentlichen Compiliervorganges, für die z.B. der Einsatz inkrementeller Methoden möglich ist, mit denen jeweils nur geänderte Teile der MNET-Beschreibung compiliert werden, liegt hier ein weites Feld für Variationen der Partitionierungsstrategien vor. Im Rahmen der vorliegenden Arbeit konnten anhand der wenigen Beispiele nur Tendenzen aufgezeigt werden, die der Vertiefung und Verifikation bedürfen. Der Aufwand dazu ist jedoch nicht zu unterschätzen, jedes der verwendeten Beispielnetze ist immerhin Gegenstand oder Teil einer eigenständigen Dissertation. Im Bereich der Compiler für Neuronale-Netze-Sprachen seien vor allem die Arbeiten von Strey [SGS96][Str99a][Str99b] und Zell [Zel94] erwähnt.

Die Tendenz der untersuchten Beispiele zeigt auf, daß zur Erreichung einer guten Lastverteilung aufgrund des unbekannten Eingangsreizes eine möglichst feinkörnige Verteilung der Neuronen einer Lage über die parallelen Rechenknoten notwendig ist. Durch ein solches Vorgehen wird jedoch ein Maximum an Verbindungsschnitten erzeugt. Die Zusammenfassung stark verknüpfter Netzbereiche führt dagegen zur Bildung von Aktivitätsschwerpunkten und somit zur Ungleichverteilung der Last.

Lastverteilung und Kommunikationsreduktion zeigen sich damit als konkurrierende Ziele. Die Benutzung der säulenbasierten geometrischen Methode hat sich in den untersuchten Beispielen als geeigneter Kompromiß erwiesen, mit dem gute Simulationsresultate erzielt wurden, so daß sie Grundlage der weiteren Betrachtungen ist.

Die Erstellung eines parallelen Simulationswerkzeugs für Workstationcluster ist jedoch nur ein Nebenprodukt des eigentlichen Ziels der vorliegenden Arbeit, nämlich der Ermittlung von Parametern für ein paralleles Spezialhardwarekonzept. Wenn auch die Simulation auf einem möglicherweise ohnehin vorhandenen lokalen Rechnernetz oder einem Parallelrechner für die Entwicklung von PCNN sehr hilfreich sein mag, so erlaubt sie nicht den echtzeitnahen Einsatz der Netze in realen Szenarien. Zum einen ist ein Parallelrechner schlicht zu groß, um ihn z.B. an einem Roboter einzusetzen, und zum zweiten erreicht die Simulationsleistung nicht die benötigten Größenordnungen für mehr als 1 Millionen Neuronen. Die Simulation des Weitzelnetzes mit 114.401 Neuronen zeigt schon beim Übergang von 4 auf 8 Rechenknoten (siehe Abb. 5-4) erste Anzeichen einer Grenze der Beschleunigung, so daß auch bei Parallelrechnern mit besserer Kommunikationsinfrastruktur bei einigen zehn Rechenknoten die maximale Geschwindigkeit erreicht wäre. Der Grund dafür liegt in der sehr kleinteiligen Kommunikation bei der PCNN-Simulation. Damit ist die benötigte zehnmal höhere Simulationsgeschwindigkeit für ein zehnmal größeres Netz nicht zu erreichen. Es besteht somit Bedarf nach einer kompakten, leistungsstarken parallelen Hardware mit einer auf das Problem abgestimmten Kommunikationsinfrastruktur.

5.4 Anforderungen an einen Accelerator

Eine solche Spezialhardware muß nach den Ergebnissen der vorgenommenen Untersuchungen verschiedene Eigenschaften aufweisen und Anforderungen erfüllen. Die vorgestellten Simulationsverfahren setzen alle einen lokalen Speicher des Rechenknotens voraus, in dem die Neuronendaten und zumindest Teile der Topologie gelagert werden können. Zudem muß der Rechenknoten die benötigte Rechenleistung zur echtzeitnahen Simulation liefern. Im Sinne eines kompakten und einfachen Systemaufbaus ist die Realisierung des lokalen Speichers als schneller on-Chip-Speicher wünschenswert. Zur Kommunikation mit Hilfe der Pre-Spikes ist ein spezielles Kommunikationssystem notwendig, da die sehr kleinen Datenpakete für Systeme mit aufwendigen Protokollen ungeeignet sind. Dabei kann auf die in modernen Kommunikationssystemen verwendete Switching-Technik zurückgegriffen werden, die mit Routing-Informationen behaftete und genormte Datenpakete durch Hardwarekomponenten ohne Einsatz von programmbasierten Protokollstapeln weiterschalten und somit auch bei vielen kleinen Paketen eine hohe Übertragungsleistung erreichen. Die verwendeten Pre-Spikes können als solche genormten Datenworte realisiert werden. Für den in der nrc-Variante eingesetzten globalen Topologiespeicher ist eine Technologie zu verwenden, die

genügende Speicherdichte mit hoher Bandbreite verbindet. Da schon am gemeinsamen Speicher festgestellt werden kann, wohin die gelesenen PostSpikes bzw. EIBs zu versenden sind, erscheint es sinnvoll, dedizierte Verbindungen zu den Rechenknoten vorzusehen. Im Sinne des einfachen und kompakten Aufbaus ist die Verwendung von Linkverbindungen mit geringer Wortbreite wünschenswert.

Aus den gewonnenen Simulationsdaten lassen sich für die Komponenten dieses Grobkonzeptes die relevanten Rahmendaten gewinnen.

- Unter der Annahme der für das Weitzelnetz verwendeten Partitionsgröße von etwa 16k Neuronen pro Rechenknoten ergibt sich der Speicherbedarf für den lokalen Speicher des Rechenknotens. Pro Neuron sind 16 bit Werte für die 4 dendritischen Potentiale, die dynamische Schwelle, den Impulsvektor, die Impulsmaske, das Inkrement der dynamischen Schwelle, die Lernschwelle, einen Eintrag in die Abklingliste und ggf. ein 32 bit breiter EIB-Cache-Eintrag vorzusehen. Der Bedarf beträgt also mindestens 20 bis 30 Byte zuzüglich diverser Statusbit. Bei 16k Neuronen ergeben sich 300 bis 400 kByte an lokalem Speicherbedarf.

- Die Rechenleistung bei entsprechender Partitionierung muß eine Berechnung von 2.204 Neuronen in der Abklingphase und 4.047 Synapsen in der Erregungsphase nahe an der Echtzeit von 1 ms pro Zeitschritt [Fra97] zulassen. Eine Simulationszeit von 1 ms pro Zeitschritt für Bildverarbeitungs-PCNN als sogenannte *Echtzeitsimulation* wird z.B. in [DDW98] postuliert. Für PCNN in der akustischen Mustererkennung wird von 0,1 ms pro Zeitschritt ausgegangen.

- Das Kommunikationsnetz für die PreSpikes muß 152 Spikes pro Zeitschritt, d.h. pro Millisekunde, und pro Rechenknoten übertragen können. Bei einem 32 bit breiten PreSpike sind das 600 kByte/s. Bei 64 Rechenknoten für 1 Mio. Neuronen entsteht ein Kommunikationsaufkommen von ca. 40 MByte/s. Dieses Aufkommen ist bei kleinteiliger Übertragung von einem reinen Bussystem nicht zu bewältigen, so daß ein Switching-Netz angebracht erscheint. Da die Kommunikation unregelmäßig erfolgen kann, sind Puffer vorzusehen.

- Der globale Topologiespeicher muß beim Weitzelnetz 2.181 Erregungen für jeden angeschlossenen Rechenknoten pro Zeitschritt aus dem Speicher lesen ($e\text{-}h_1$). Bei einer 32 bit breiten EIB-Zeile sind das etwa 10 MByte/s pro Rechenknoten. Diese Datenmenge muß zudem über die Linkverbindungen zu den Rechenknoten übertragen werden und dort entgegengenommen werden. Der gesamte Speichertransfer bei 64 Rechenknoten würde mit 640 MByte/s die Möglichkeiten moderner DRAM-Speicher sehr weit ausreizen, so daß von mehreren Topologiespeichern auszugehen ist.

- Werden die ca. 4 Mio. Verbindungen des Weitzelnetzes hochgerechnet auf 1 Mio. Neu-

ronen, so entsteht für 32 Mio. Verbindungen mit 32 bit pro Verbindung ein Speicherbedarf von 128 MByte. Dazu kommt die gleiche Speichermenge, wenn zu Lernzwecken eine empfängerorientierte Topologiespeicherung benötigt wird, sowie Speicher für die globale Spikeliste und den Blockstartspeicher. DRAM-Speicher im Umfang von einigen hundert MByte läßt sich unproblematisch und kompakt in Form weniger Speichermodule bereitstellen. Schneller SRAM-Speicher scheidet hingegen bei diesen Größenordnungen aus.

- Für das Brausenetz empfiehlt sich eine Simulation mit der rc-Simulator-Variante. Es kann dann auf einen globalen Topologiespeicher verzichtet werden, da sich die 12 rc-Masken mit zusammen 378 Verbindungen (siehe Tab. 5-2) ohne Probleme in dem ansonsten für den EIB-Cache vorgesehenen lokalen Speicher des Rechenknotens unterbringen lassen.

Diesen Anforderungen entsprechend wird im folgenden ein Konzept für eine Spezialhardware vorgestellt und bezüglich seiner Leistung abgeschätzt.

Kapitel 6 - Konzept einer Spezialhardware auf Basis von DSPs

Das Ziel der vorliegenden Arbeit ist die Konzeption eines einfachen und kompakten Systems zur echtzeitnahen Simulation großer PCNN. Dieses Ziel ist mit der Anforderung konkretisiert worden, 1 Mio. Neuronen mit einer Geschwindigkeit von 1 ms pro Zeitschritt zu simulieren. In den vorangegangenen Kapiteln wurden Untersuchungen vorgestellt, die eine Lösung der Aufgabenstellung durch ein paralleles System von kommerziellen Mikroprozessoren möglich erscheinen lassen, wenn verschiedene ermittelte Randbedingungen eingehalten werden. Inhalt dieses Kapitels ist nun die Vorstellung eines Systemvorschlags, der diese Randbedingungen weitgehend einhält.

Als Rechenknoten für das System ist ein digitaler Signalprozessor (DSP) mit einem großen internen Speicher und einem Kommunikationssystem mit Linkverbindungen und Multiprozessorunterstützung ausgewählt worden. Dieser Prozessor bzw. diese Prozessorfamilie wird beschrieben und eingeordnet. Für die PreSpike-basierte Kommunikation ist ein System von hierarchischen Bussen konzipiert worden, das über Kommunikationseinheiten in Switching-Technik verbunden ist. Für die Kommunikationseinheiten wird eine Implementierung in Form von programmierbarer Logik auf Basis von FPGAs (Field Programmable Gate Array) vorgestellt. Für ein gemeinsames Speichersubsystem wird eine ebenfalls auf FPGAs sowie auf SDRAM-Speichern beruhende Realisierung vorgeschlagen. Auf Basis dieser Komponenten sind zwei Modultypen entworfen worden, von denen ein Typus das rc-Simulationsverfahren mit verteilter Topologiespeicherung realisiert, während der andere um das Speichersubsystem ergänzt Berechnungen nach dem nrc-Modell mit gemeinsamer Topologiespeicherung zuläßt. Für die Module ist eine Realisierung in Form von VME-Bus-Platinen entwickelt worden, sowie ein Alternativvorschlag auf Basis des PCI-Busses. Auf Basis der VME-Bus-Platinen lassen sich aus den Modulen in Kombination mit einer VME-Bus-Sun-Workstation Systeme erstellen, welche die Anforderungen für einige Millionen Neuronen zumindest annähernd erfüllen. Das Konzept wird im anschließenden Kapitel aufgrund der gewonnenen Daten und weiterer Untersuchungen in seiner Leistung abgeschätzt und in den Kontext anderer Konzepte und Realisierungen von PCNN-Simulatoren gestellt.

6.1 Systemaufbau

Orientiert an der Zielvorgabe der Simulation von 1 Mio. Neuronen wird im weiteren ein Prototypsystem betrachtet, das aus zwei nrc-Modulen und einem rc-Modul sowie der VME-Bus-Sun besteht (siehe Abb. 6-1). Ein solches System ist ähnlich kompakt wie ein handelsüblicher PC und mit den in Versuchsaufbauten üblichen VME-Modulen kombinierbar, so daß wesentliche Anforderungen erfüllt werden (siehe Kap. 3.5).

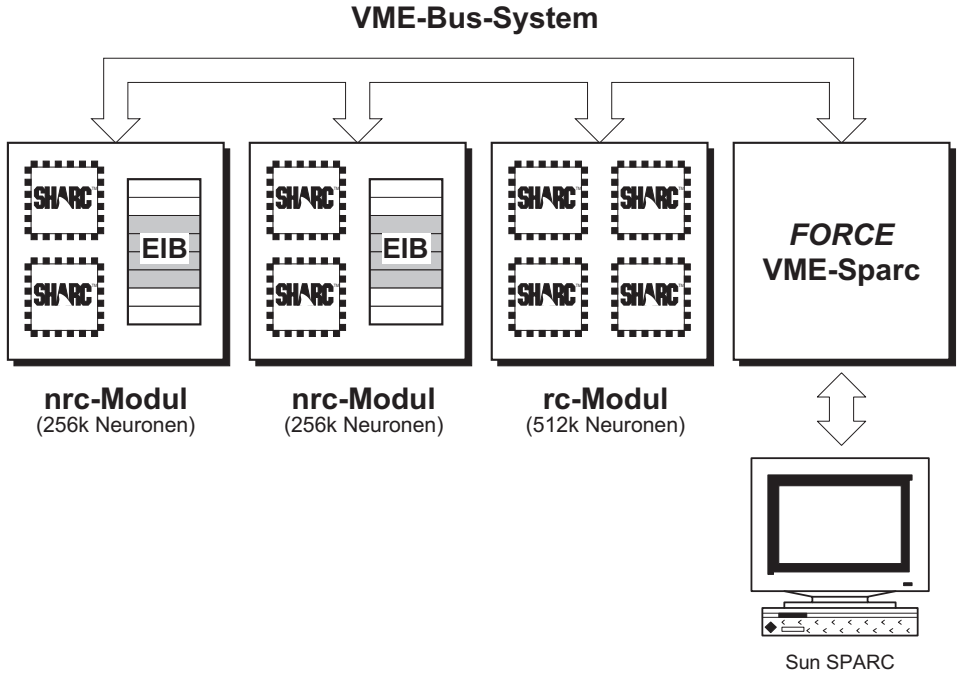

Abb. 6-1 Systemvorschlag für ein Prototypsystem zur Simulation von 1 Millionen Neuronen auf Basis des VME-Busses

Für ein nrc-Modul ist die Simulation von Teilnetzen mit bis zu 262.144 Neuronen (256k) vorgesehen, die aufgrund des vorhandenen globalen Topologiespeichers mit nahezu beliebigen Verknüpfungsstrukturen verbunden werden können. Auf diesen Modulen ist auch der Einsatz von Lernverfahren vorgesehen. Die rc-Module sollen in der Lage sein, bis zu 524.288 Neuronen (512k) zu simulieren. Diese dürfen nur mit Hilfe von EIB-Masken (rc-EIBs, siehe Kap. 4.3) verknüpft sein, die im lokalen Speicher der Rechenknoten untergebracht werden, so daß ein Speichersubsystem entfallen kann und stattdessen die doppelte Anzahl von Rechenknoten integriert wird. Wie im weiteren gezeigt wird, ist der lokale Speicher des Rechenknotens nicht ausreichend dimensioniert, um dort die Simulatorvariante mit verteilter Topologiespeicherung auf Basis von normalen EIBs durchzuführen, so daß nur die rc-Variante auf Basis von rc-EIBs betrachtet wird. Das ganze System wird von der Sun-VME-Bus-Workstation gesteuert, die während der Simulation externe Spikes einspeist und interne Spikes protokolliert. Insgesamt besteht das System damit aus vier VME-Modulen.

Die *rc-Module* erlauben die höchste Dichte von Rechenknoten, da sie ansonsten nur noch aus dem Kommunikationssystem für die PreSpikes und einer VME-Bus-Schnittstelle bestehen (Abb. 6-2). Auf VME-Bus-Modulen der Größe 220 mm x 233 mm lassen sich bei beidseitiger Bestückung 32 der ausgewählten Signalprozessoren (DSP) unterbringen.

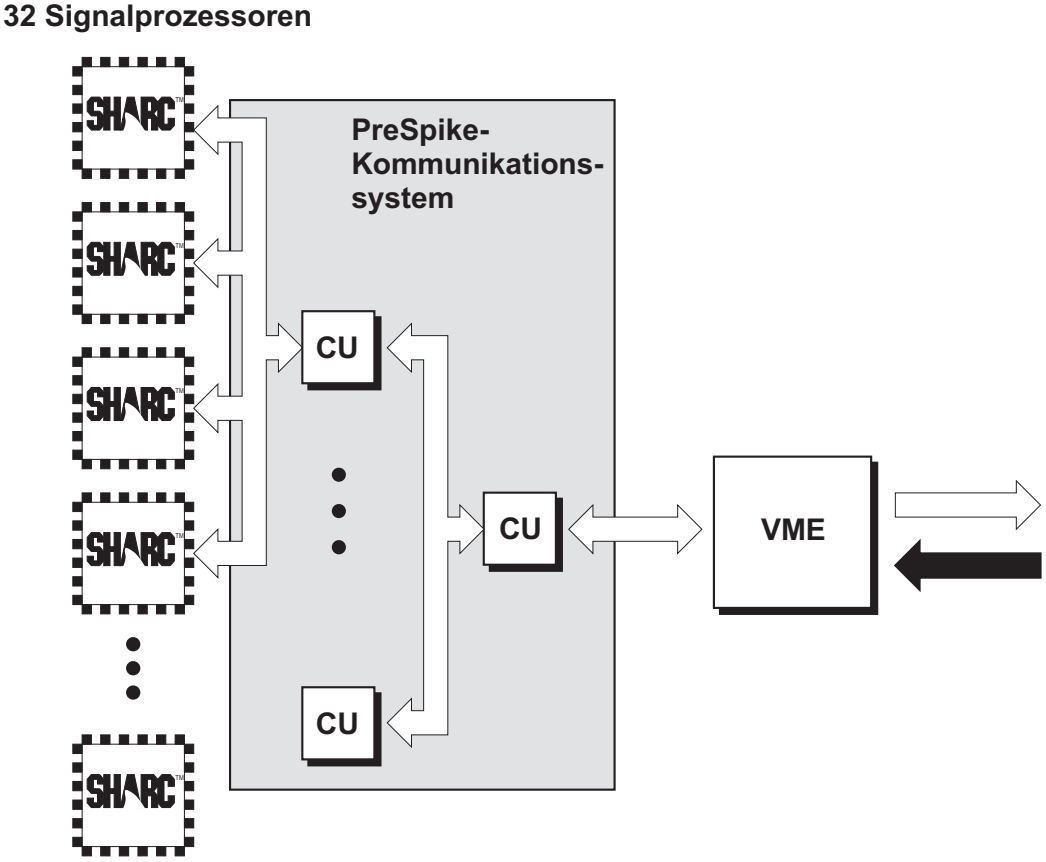

Abb. 6-2 Aufbau eines rc-Moduls für den DSP-Parallelrechner

Als Rechenknoten wurde der ADSP21060 der Firma Analog Devices [AD96] bzw. alternativ einer der kompatiblen Nachfolgeprozessoren aus der SHARC-Familie [AD00] gewählt. Diese verfügen von Haus aus über eine multiprozessorfähige Busschnittstelle, über die je vier der Prozessoren mit einer Kommunikationseinheit (CU) an einem Bus betrieben werden. Die CUs werden wiederum zusammen an einem Bus betrieben, so daß ein hierarchisches Bussystem entsteht, in welchem die DSPs die Blätter bilden und die VME-Bus-Schnittstelle als Wurzel dient. Über das CU-System erfolgt die Kommunikation mittels PreSpikes zwischen den DSPs untereinander und zwischen dem VME-Bus und den DSPs in beiden Richtungen. Auf einem rc-Modul kommen 11 CUs in drei Hierarchiestufen zum Einsatz, die im Rahmen der Diplomarbeiten [Joc00][Str00] in Designs für insgesamt 5 FPGAs umgesetzt wurden. Die CUs sind als modulare Designs entworfen, die aus einem einheitlichen Kern in Form eines PreSpike-Switches mit Puffern sowie aus verschiedenen Schnittstellenmodulen zur Verbindung mit den DSPs, dem VME-Bus und anderen CUs bestehen.

Die *nrc-Module* basieren ebenfalls auf einem CU-System, das allerdings nur 16 DSPs zu bedienen hat und zudem die PreSpikes nur von den Blättern zur Wurzel übertragen muß. An dieser Wurzel befindet sich jedoch neben der VME-Bus-Schnittstelle auch das Speichersubsystem für den gemeinsamen Topologiespeicher (siehe folgende Abb. 6-3).

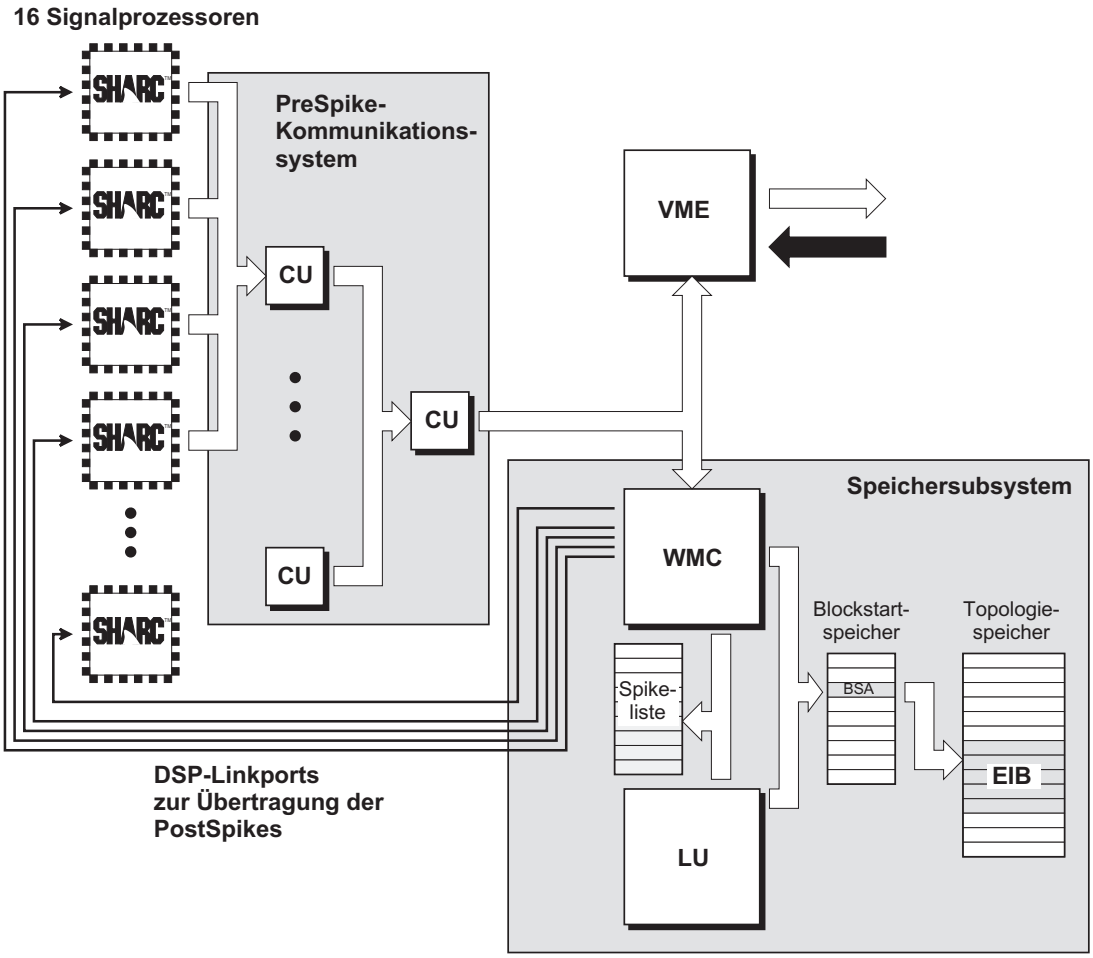

Abb. 6-3 Aufbau eines nrc-Moduls für den DSP-Parallelrechner

Das Speichersubsystem beinhaltet einen Topologiespeicher für eine senderorientierte und eine empfängerorientierte Ablage der EIBs, den Blockstartspeicher zur Adressierung sowie die globale Spikeliste. Gesteuert wird das System durch einen *Weight Memory Controller* (WMC). Daneben kann optional eine Lerneinheit (LU) integriert werden, die auf der Netztopologie die in [SH99][Sch00][SSW00] beschriebenen Verfahren anwendet. In der Abklingphase empfängt der WMC von den lokalen DSPs und vom VME-Bus PreSpikes und trägt sie in die globale Spikeliste ein. In der Erregungsphase werden die zu den PreSpikes gehörenden EIBs gelesen und die PostSpikes werden den Zielen entsprechend über die Linkverbindungen der DSPs an diese verteilt. Sind keine Linkverbindungen vorhanden, kann alternativ das PreSpike-Kommunikationssystem benutzt werden, da nur in der Abklingphase dort eine Übertragung stattfindet und die PostSpikes in der Erregungsphase transferiert wer-

den. Da der WMC in der Abklingphase nicht auf den Topologiespeicher zugreift, kann die LU diesen zu Lernzwecken modifizieren. Der WMC und das Speichersubsystem wurden in der Diplomarbeit [Ull00] in Form eines FPGAs und mehrerer SDRAM-Riegel entworfen.

Neben der PreSpike-Übertragung dienen die CUs auch zur Synchronisation der Phasen- und Zeitschrittwechsel. Diese Synchronisation erfolgt zudem über den VME-Bus, so daß alle DSPs zeitgleich die Phase und den Zeitschritt wechseln. Neben der *Simulationsphase*, in der die Module weitgehend unabhängig die gewünschte Anzahl von Zeitschritten berechnen, ist eine *Initialisierungsphase* vorgesehen, in der die VME-Bus-Sun über den Bus vollständigen Zugriff auf die Speicher der Module inklusive der lokalen Speicher der Rechenknoten hat.

In den folgenden Abschnitten wird zunächst der ausgewählte Rechenknoten vorgestellt und seine Wahl wird begründet. Daran anschließend werden die CUs und mit ihnen das PreSpike-Kommunikationssystem und die Phasensteuerung beschrieben. Weiterhin erfolgt die Darstellung des WMCs und des mit ihm verbundenen Speichersubsystems. Das Kapitel schließt mit der Präsentation einiger Realisierungsvorschläge und einer Zusammenfassung der Eigenschaften der entwickelten FPGAs. Das gesamte Hardwarekonzept trägt in Anlehnung an den SPIKE128K den Namen *ParSPIKE* [WHR99a][WHR99b][SSW00].

6.2 Auswahl eines geeigneten Prozessors

Bei der Auswahl eines geeigneten Rechenknotens ergeben sich verschiedene Anforderungen, die gewöhnlich nicht alle zugleich erfüllt werden können, so daß ein Kompromiß zu ermitteln ist. Die in Kap. 3.5 und Kap. 5.4 formulierten Anforderungen an den Rechenknoten eines parallelen PCNN-Simulators sollen wie folgt noch einmal zusammengefaßt werden.

- Die *Rechenleistung* muß ausreichen, um die Berechnungen der Abkling- und Erregungsphase für einen Zeitschritt in einer Zeit *nahe an 1 ms* durchzuführen [Fra97] [DDW98]. Der zu bewältigende Rechenaufwand hängt dabei von der Anzahl der auf den einzelnen Knoten verteilten Neuronen ab.
- Der *lokale Speicher* muß ausreichend dimensioniert sein, um die Neuronendaten und die jeweils benötigten Topologiedaten der Simulationsverfahren (siehe Kap. 5.4) sowie den Algorithmus selbst unterzubringen. Die tatsächlich auf einen Knoten verteilbare Neuronenanzahl hängt also im wesentlichen von den Speicher- und Rechenleistungsressourcen ab. Eine größere Anzahl von Knoten zur Erreichung der Gesamtzahl der zu simulierenden Neuronen führt jedoch zu einem aufwendigeren Systemaufbau und zu einer feinkörnigeren Parallelisierung, so daß eine möglichst hohe Anzahl von Neuronen pro Rechenknoten anzustreben ist.
- Die *Kommunikationsressourcen* sollten ausreichend dimensioniert sein, um die Übertra-

gung der PreSpikes bzw. den Empfang der PostSpikes durchführen zu können. Zudem sollten alle Komponenten zum Aufbau eines Multiprozessorsystems vorhanden sein.

- Der Rechenknoten sollte in jeder Hinsicht *effizient* sein, d.h. die Anforderungen sollten mit geringem Aufwand an Platz, Kosten und Energiebedarf erreicht werden. Zur Effizienz gehört auch eine einfache Implementierbarkeit mit niedrigem Testaufwand und einer guten Unterstützung der Entwicklung, so daß der Zeitaufwand der Implementierung gering bleibt. Auch die Wartbarkeit, die Flexibilität bei Änderungen und Erweiterungen sowie die Benutzerfreundlichkeit sind diesem Kriterium zuzuordnen.

Insbesondere der letzte Punkt war der Grund für die vorliegende Arbeit mit dem Ziel, auf die Entwicklung einer speziellen Prozessoreinheit - sei es als FPGA-basierte Lösung wie beim SPIKE128k [Fra97] oder als ASIC (Application Specific Integrated Circuit) beim NESPINN/MASPINN-Projekt [JRK96][SMJ98] - zu verzichten. Stattdessen sollen die Anforderungen mit einem kommerziellen Prozessor erfüllt werden, auf dessen Basis ein paralleles System die benötigte Leistung erreicht. Wünschenswertes Ziel vor dem Hintergrund der Effizienz ist der Einsatz eines Rechenknotens, der alle Eigenschaften auf einem Chip integriert. Die Auswahl und Bewertung eines solchen Chips ist u.a. in zwei Studienarbeiten [Rah99][Sil00] und einer Diplomarbeit [Rah00] erfolgt. In diesen Arbeiten wurde der ausgewählte DSP zudem programmiert und getestet.

Grundsätzlich kann die Auswahl eines Rechenknotens aus den drei Grundtypen von hochintegrierten digitalen Rechenchips erfolgen, deren Einteilung jedoch recht fließend ist.

- *Microcontroller* sind auf Kontrollaufgaben spezialisierte Prozessoren. Sie werden häufig zur Steuerung elektronischer Geräte eingesetzt, weshalb sie über umfangreiche Schnittstellen (I/O-Ressourcen) verfügen und weitgehend ohne zusätzliche Peripheriebausteine auskommen. Die reine Rechenleistung steht dabei oft im Hintergrund, während Speicher auf dem Baustein (on-Chip) gewöhnlich integriert ist. Da Microcontroller als „embedded“ Komponenten eingesetzt werden, ist die Effizienz unter Energiebedarfs-, Kosten- und Platzaspekten ein wichtiges Optimierungsziel.

- *Digitale Signalprozessoren* (DSP) sind hingegen meist auf den Rechenfluß optimiert, da sie z.T. aufwendige Berechnungen auf großen Datenmengen ausführen. Die Rechenwerke sind daher mit schnellen Einheiten (z.B. Multiplizierern) ausgestattet, die eine Abarbeitung der meisten Befehle in einem Takt zulassen. Die Kommunikationsschnittstellen sind auf den schnellen Transfer großer Datenmengen vom und zum Baustein ausgelegt. Häufig ist auf dem Baustein on-Chip-Speicher integriert.

- *Mikroprozessoren* kommen in Standardrechnern zum Einsatz und sind gewöhnlich auf die Abarbeitung sequentieller Kontrollflüsse optimiert. Im Gegensatz zu den Microcontrollern werden jedoch erhebliche Peripheriekomponenten zum Aufbau eines vollständi-

gen Rechnersystems benötigt. Dafür ist die Rechenleistung zumeist höher. Die vorhandenen on-Chip-Speicher können vom Programmierer nicht beliebig genutzt werden, da sie als Cache-Speicher ausgelegt sind. Die Rechenwerke benötigen meist mehrere Takte für eine Operation, allerdings sind die Mikroprozessoren wesentlich höher getaktet als die DSPs oder Microcontroller.

Viele der Charakteristika der einzelnen Gruppen tauchen mittlerweile auch in den anderen Gruppen auf. Allerdings haben die Untersuchungen u.a. in [Rah99][Sil00] ergeben, daß zumindest zum Zeitpunkt der Erstellung dieser Arbeit nur eine Gruppe von Prozessoren für den Rechenknoten des ParSPIKE in Frage kommt. Die verfügbaren Microcontroller sind aufgrund ihrer Rechenleistung nicht in der Lage, eine genügende Neuronenanzahl zu simulieren. Bei den konventionellen Mikroprozessoren ist insbesondere der on-Chip-Speicher nicht nutzbar, da er als Cache eingesetzt wird, so daß der Aufbau eines ParSPIKE-Rechenknotens den Einsatz einer Vielzahl weiterer Komponenten notwendig machen würde. Im Bereich der DSPs konnte hingegen die Gruppe der Hochleistungssignalprozessoren ausgemacht werden, die sowohl die benötigten on-Chip-Ressourcen als auch eine hohe Rechenleistung bereitstellen und damit die wichtigsten Eigenschaften auf einem Baustein integrieren.

Aus dieser Gruppe wurden die DSPs der Analog Devices SHARC-Familie (Super Harvard Architecture Computer) [AD96][AD00][Sil00] in erster Linie wegen ihres großen on-Chip-Speichers ausgewählt. Der erste Vertreter der Familie, der ADSP21060 (siehe Abb. 6-4), auf dem die Verfahren implementiert wurden, verfügt über 512 kByte SRAM auf dem Chip.

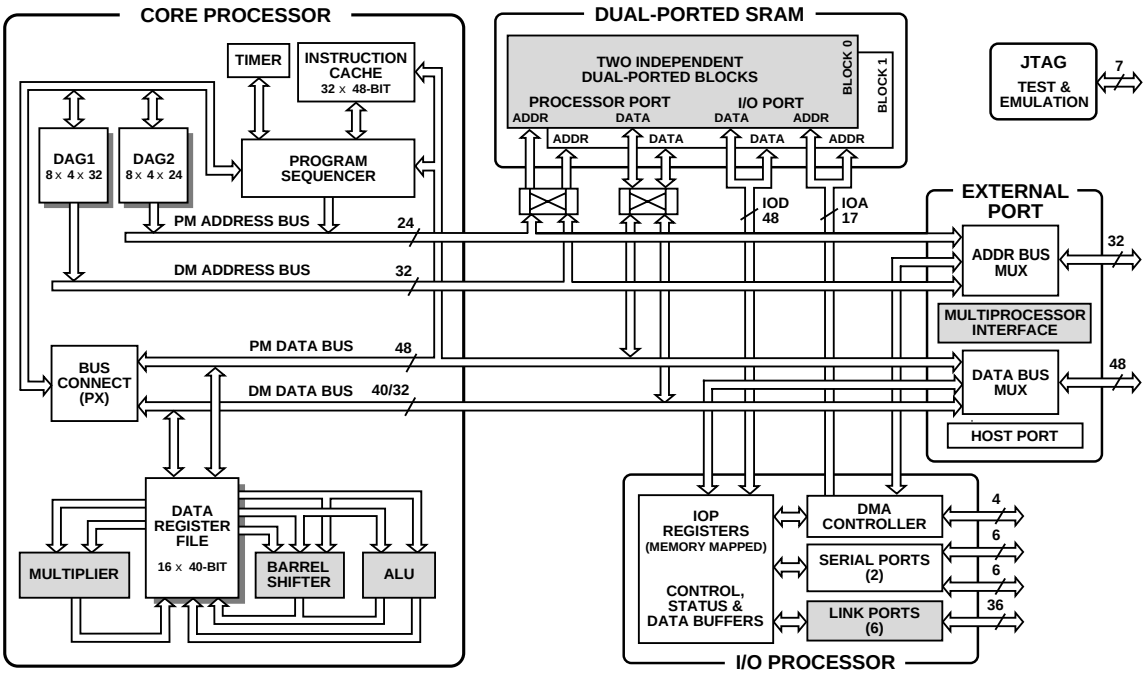

Abb. 6-4 Analog Devices SHARC-DSP-Architektur (ADSP21060) [AD96]

Der ADSP teilt sich in vier wesentliche Blöcke:

- Der sogenannte *Core Processor* führt die Berechnungen durch. Dazu stehen ihm drei Fließkommaeinheiten in Form eines Multiplizierers, einer arithmetisch-logischen Einheit (ALU) und einer Schiebeeinheit (Shifter) zur Verfügung. Diese Einheiten können gleichzeitig jeweils einen Befehl pro Takt abarbeiten und dazu ebenfalls gleichzeitig auf einen gemeinsamen Registerblock zugreifen. Der Prozessorkern wird durch eine Programmsteuerung mit zwei Adreßgeneratoren ergänzt, so daß indirekte Adressierungen die ALU nicht belasten. Mit einem Takt von 40 MHz für den ADSP21060 ergibt sich eine theoretische Spitzenleistung von 120 MFLOPs (Million Floating Point Operations).
- Der *on-Chip-Speicher* des ADSP ist in Form von zwei Blöcken zu 256 kByte angelegt, auf die jeweils zwei Einheiten zugleich zugreifen können (dual-ported). Somit können zwei Operanden zugleich aus dem Speicher geholt werden, während von außen Transfers in und aus dem Speicher ablaufen.
- Der *I/O Processor* wickelt solche Transfers in den Prozessor hinein und aus ihm heraus ohne Belastung des Prozessorkerns ab. Auch komplexe Zugriffe lassen sich so im DMA (Direct Memory Access)-Modus abwickeln. Nach Abschluß des Transfers wird der Prozessorkern per Interrupt benachrichtigt. Der Schnittstellenprozessor verwaltet insbesondere auch die Linkschnittstellen des ADSP, von denen eine ggf. zum Transfer der PostSpikes vom WMC zum ADSP benutzt wird. Diese Linkschnittstelle ist 4 bit breit und kann bei einem Takt von 40 MHz bis zu 20 MByte/s übertragen, wobei der Übertragungstakt von der übertragenden Komponente vorgegeben wird.
- Der *External Port* verfügt über eine Multiprozessor-Unterstützung, mit der bis zu sieben Teilnehmer an einem gemeinsamen Bus betrieben werden können. Dazu ist ein Zugriffsprotokoll und eine Busarbitrierung implementiert. Der Bus ist 48 bit breit und wird mit 40 MHz betrieben. Zum Anschluß langsamerer Komponenten ist ein asynchrones Protokoll vorgesehen.

Auf dem ADSP21060 ist das nrc-Simulationsverfahren und das rc-Verfahren implementiert. Der Speicherblock dient dabei zur Unterbringung der Neuronendaten, des EIB-Caches bzw. rc-EIB-Speichers und des Simulationsprogramms. Über die Multiprozessor-Schnittstelle werden die PreSpikes ausgegeben. In der rc-Variante werden über diese Schnittstelle zudem auch PreSpikes empfangen. In der nrc-Variante werden stattdessen die nicht im EIB-Cache befindlichen PostSpikes über die Link-Schnittstelle empfangen.

Im Rahmen von [Rah99][Rah00] sind die C-Programme, die Grundlage der Simulationen aus Kap. 3.5 waren, zunächst auf den DSP umgesetzt worden. Kritische Programmteile sind dann schrittweise in Assembler codiert und optimiert worden. Die Neuronendaten, die Ab-

klingliste und der EIB-Speicher werden im internen Speicherblock des ADSP abgelegt. Insgesamt werden 448 kByte der vorhandenen 512 kByte zur Unterbringung der Daten für 16.384 Neuronen (16k) benötigt. Da unter anderem die verbliebenen 64 kByte des internen Speichers nicht zur Unterbringung des Programms und eines Betriebssystems ausreichen, ist auf ein Betriebssystem verzichtet worden, was unproblematisch ist, da zur Simulation nur ein einziges Programm auf dem ADSP eingesetzt wird. Im Rahmen dieses Programmes sind damit die Routinen für den Speicherzugriff und die Kommunikation zu implementieren. Der Speicherort der Neuronendaten ist fest vorgegeben und aus einem Offset und der Neuronenadresse des Neurons gebildet, so daß von außen (von der VME-Bus-Sun) ohne Umweg auf die Neuronendaten zugegriffen werden kann. Für jeden Neuronenparameter sind im Simulatorprogramm Zugriffsroutinen implementiert, mit denen während der Simulation unter Angabe der Neuronenadresse auf den jeweiligen Parameter zugegriffen wird. Da bei Analog Devices ein erklärungsbedürftiger Unterschied zwischen einem „native support“ und einem „support“ für 16 bit Datenworte besteht, die meisten Neuronendaten jedoch 16 bit breit sind, mußte innerhalb der Zugriffsroutinen auch das Maskieren und Schieben der Datenworte erfolgen, um so zwei 16 bit Worte in einem 32 bit Wort unterzubringen. Dabei kommt die Shifter-Einheit des ADSP zum Einsatz. Auf die trotzdem zeitrelevanten Auswirkungen der fehlenden 16 bit Unterstützung wird im weiteren noch eingegangen. In den Neuronendaten befindet sich für die rc-Variante zudem genügend Speicher, um von bis zu 16k externen Neuronen PreSpikes aufnehmen und mit ihnen rc-EIBs adressieren zu können. Die Unterbringung der Datenstrukturen zur Kommunikation mit allen Neuronen des Netzes ist im Gegensatz zu den PVM-Simulationen aufgrund des Speicherbedarfs nicht möglich.

Die Kommunikation wird über den Schnittstellenprozessor abgewickelt. Der Versand von PreSpikes über die Multiprozessorschnittstelle erfolgt durch die Initiierung eines DMA-Transfers, der unabhängig vom Kernprozessor durch den Schnittstellenprozessor abgearbeitet wird. Der PreSpike-Empfang in der rc-Variante erfolgt durch einen von außen (von der CU) initiierten DMA-Transfer. Nach dem Empfang von 16 PreSpikes wird von einer Interrupt-Service-Routine ein Semaphor gesetzt, der regelmäßig vom Hauptprogramm abgefragt wird, so daß dieses die PreSpikes in die lokale Spikeliste übernehmen kann. Ebenso erfolgt der Empfang der PostSpikes über die Linkports. Für die Synchronisation wird ein Message-Register des ADSP verwendet, das ebenfalls mit einer Interrupt-Routine verbunden ist. Bei Abschluß einer Phase setzt der ADSP ein Bit in einem Register der CU. Haben alle vier Prozessoren das Bit gesetzt und hat die CU die Bestätigung, daß alle anderen ADSPs auch die Phase abgeschlossen haben, so setzt sie mit einem Broadcastzugriff die Message-Register der angeschlossenen ADSPs zurück, worauf diese mit dem Eintritt in die nächste Phase reagieren. Auf die gleiche Weise erfolgt der Zeitschrittwechsel.

Die Verwendung der kommerziellen Signalprozessoren der SHARC-Reihe bietet den Vorteil, von der Weiterentwicklung dieser Prozessoren profitieren zu können. Aktuell befindet

sich der Nachfolgeprozessor des verwendeten ADSP21060 - der ADSP21160 - auf dem Markt, eine Weiterentwicklung in Form des TigerSHARC befindet sich vor der Markteinführung [AD00]. Neben diesen Prozessoren existiert eine ganze Reihe von Abwandlungen, bei denen insbesondere Speicher und Kommunikationsmöglichkeiten beschnitten wurden. Der ADSP21160 und TigerSHARC sind Code- und Protokoll-kompatibel zum ADSP21060, so daß sie ohne Probleme als Ersatz im Konzept angesehen werden können. In Tabelle 6-1 sind einige der wichtigsten Parameter dieser Prozessoren angegeben. Zur Ermittlung der Simulationsgeschwindigkeit für das Weitzelnetz sei auf Kap. 7 verwiesen.

	ADSP21060	ADSP21160	TigerSHARC TS001	TigerSHARC TSxxx
on-Chip Speicher	512 kByte	512 kByte	768 kByte	768 kByte
Taktfrequenz	40 MHz	100 MHz	150 MHz	250 MHz
Simulationszeit für das Weitzelnetz (8PE) pro Zeitschritt	25 ms	10 ms	6,7 ms	4 ms
Energieverbrauch	1,5 - 2 W	3 - 4 W	n.e.	n.e.
Kernspannung	3,3 V	2,5 V	2,5 V	n.e.
Abmessungen	35 x 35 mm	27 x 27 mm	n.e.	n.e.

Tab. 6-1 Die Prozessoren der SHARC-Familie [AD96][AD00]

Insbesondere die Geschwindigkeitsabschätzung für den ersten Vertreter der Familie - den ADSP21060 - zeigt, daß die gewünschte Simulationsgeschwindigkeit nicht erreicht wird. Gegenüber den ersten Abschätzungen, die zur Auswahl des Prozessors für das Konzept führten [Rah99], hat insbesondere die fehlende 16 bit Unterstützung zu Zusatzaufwand geführt, der etwa ein Drittel des gesamten Aufwands verursacht [Rah99]. Die für den ADSP21060 untersuchte Umsetzung der Verfahren wurde für die Nachfolgeprozessoren mit dem Prozessortakt skaliert, was realistisch ist, da alle zeitrelevanten Elemente interne Prozessorkomponenten sind, die ohne Latenztakte auskommen [AD00]. Durch Umstellungen des Programmcodes kann ggf. davon profitiert werden, daß die Nachfolgeprozessoren über ein zweites Rechenwerk verfügen. Die beiden Rechenwerke können allerdings nur im SIMD-Betrieb genutzt werden [AD00], so daß sicher keine Halbierung der Simulationszeit erreicht wird. Weitere Beschleunigung ist durch den „native support“ von 16 bit Zugriffen durch die TigerSHARC-Modelle [AD00] zu erwarten, so daß dort eine weitere Simulationszeitreduktion von etwa 30% angesetzt werden kann. Damit ist zumindest in der Perspektive die gewünschte Simulationszeit mit solchen kommerziellen Prozessoren erreichbar.

Durch die Integration aller Rechenknotenkomponenten auf dem Prozessorchip läßt sich, wie nachfolgend noch gezeigt wird, auch die Bestückung der ParSPIKE-Module mit bis zu 32

Prozessoren erreichen. Da der interne Speicher zur Unterbringung von 16k Neuronen ausreicht, wird somit die gewünschte Gesamtneuronenzahl pro Modul ebenfalls erreicht. Ein weiterer relevanter Punkt ist der Energiebedarf solcher Module, der die Einsatzmöglichkeiten in einem VME-Bus-System begrenzen kann. Da während der Simulation von einer überwiegenden Vollauslastung der Prozessoren ausgegangen werden kann, muß hier der maximale Bedarf angenommen werden. Für den ADSP21060 würden bei 32 Prozessoren allein von diesen 64 W benötigt, was in VME-Bus-Systemen noch unkritisch ist. Die 128 W beim Einsatz des ADSP21160 erfordern sicherlich zusätzliche Spannungsversorgungsmaßnahmen. Für den TigerSHARC sind entsprechende Werte noch nicht erhältlich, es ist jedoch mit einem weiter erhöhten Energiebedarf zu rechnen.

Insgesamt zeigt sich, daß die SHARC-DSPs in wesentlichen Belangen den Anforderungen an einen Rechenknoten genügen. Die hohe Integration und der große Speicher der Prozessoren erlauben den gewünschten kompakten Aufbau eines Prototypsystems. Durch die Abbildung wesentlicher Teile der Simulation in Software und die Unterstützung der Entwicklung durch entsprechende Werkzeuge [AD96][AD00][Rah99][Rah00] wird ein hohes Maß an Flexibilität bezüglich Änderungen am Verfahren und eine vereinfachte Test- und Implementierbarkeit erreicht. Der Zugriff auf sämtliche internen Speicher von der VME-Bus-Sun unterstützt Wartung und Handhabung. Die benötigte Rechenleistung für die aufgrund der sonstigen Randbedingungen simulierbare Anzahl von Neuronen wird jedoch nicht bereitgestellt und ist erst im Verlauf der weiteren Entwicklung der Baureihe zu erwarten. Durch eine Absenkung der Neuronenanzahl pro Rechenknoten würden wiederum die sonstigen Randbedingungen verfehlt, so daß dieser Weg zur Erreichung der Simulationszeiten ausscheidet.

Statt der SHARC-Prozessoren lassen sich auch andere Prozessoren für das Verfahren einsetzen. Ein on-Chip-Speicher kann dabei durch externes SRAM substituiert werden, eine fehlende Multiprozessor-Unterstützung durch eine aufwendigere Kommunikationseinheit (CU). Auch auf einen Linkport kann ggf. verzichtet werden, wenn der PostSpike-Transfer auf die CU verlagert wird. Allerdings führt eine geringere Integration der Rechenknotenkomponenten zu einem erhöhten Entwicklungsaufwand und Ressourcenbedarf - insbesondere bezüglich der Platzanforderungen. Dabei ist das Vorhandensein von internem Speicher vor dem Hintergrund von etwa 130.000 Speicherzugriffen pro Zeitschritt bei 16k Neuronen pro Rechenknoten (siehe Kap. 7.3) wesentlich. Über einen genügend großen on-Chip-Speicher verfügte zu Beginn der Konzepterstellung nur der ADSP21060 [Rah99]. Mittlerweile sind in Form des Lucent DSP16410 oder des Texas Instruments TMS320C62x weitere Hochleistungs-DSPs mit großem on-Chip-Speicher erhältlich bzw. angekündigt. Auch im Bereich der Microcontroller zeichnet sich durch Verwendung von embedded DRAM eine solche Entwicklung z.B. in Form des Mitsubishi M32/RD ab [Sil00]. Der Einsatz von Alternativen zu den SHARC-Prozessoren wurde beim Systementwurf soweit möglich berücksichtigt. Ein Vergleich mit einigen Beispielen und eine Wertung sind in Kap. 7.5 zu finden.

6.3 Kommunikationssystem

Von einem konventionellen Parallelrechnersystem und auch von universell einsetzbaren DSP-Parallelrechnern unterscheidet sich das vorgestellte Konzept insbesondere durch ein spezielles, auf Hardwarekomponenten basierendes Kommunikations- und Synchronisationssystem. Die Kommunikationseinheiten (CU) ermöglichen die Abwicklung der sehr kleinteiligen PreSpike-Übertragung ohne allzu große Latenzzeiten. Gerade die Tatsache, daß bei der Kommunikation mittels PreSpikes einzelne Datenworte zu übertragen sind, führt insbesondere beim Einsatz von programmbasierten Protokollstapeln oder komplexen Busprotokollen zu einer geringen Übertragungsrate aufgrund der großen Latenzzeiten. Diesem Problem tritt das CU-Netzwerk entgegen, indem die Übertragung durch ein schaltungstechnisch realisiertes Routing stattfindet. Dazu ist für den PreSpike ein einheitliches 32 bit Datenformat definiert worden [Rah99][Rah00], das die benötigten Routinginformationen enthält und eine Zielfeststellung durch die CUs über einen einfachen bitweisen Vergleich ermöglicht.

Je nach Modultyp treten verschiedene Übertragungsmodi für die PreSpikes auf. Neben der Neuronenadresse des feuernden Neurons werden dem PreSpike noch eine DSP-Kennung, eine Boardkennung für das Modul, ein Hostbit sowie verschiedene Statusbits hinzugefügt. Im Fall eines *nrc-Moduls* müssen die PreSpikes grundsätzlich nur von den Blättern (DSPs) zur Wurzel (VME-Bus bzw. WMC) übertragen werden, da den DSPs externe Erregungen über den WMC als PostSpikes zugeführt werden. Auf einem solchen Modul ergeben sich damit die folgenden Übertragungsarten.

1. Die Übertragung *vom lokalen DSP zum WMC* erfolgt für solche PreSpikes, deren Zielneuronen wiederum auf lokalen DSPs liegen. Da der WMC zur Adressierung des EIBs die genaue Identifikation des feuernden Neurons benötigt, werden als Board- und DSP-Kennung diejenigen des feuernden Neurons eingetragen.

2. Die Übertragung *vom lokalen DSP zu einem DSP auf einem anderen nrc-Modul* erfolgt durch Versand über die VME-Bus-Schnittstelle zum WMC des Zielboards. Dazu versieht der DSP den PreSpike mit der Boardkennung des Zielboards. An der VME-Bus-Schnittstelle wird diese Kennung auf diejenige des sendenden Boards zurückgesetzt und der PreSpike wird an den WMC des Zielboards übertragen, der mit ihm direkt den EIB adressieren kann.

3. Die Übertragung *vom lokalen DSP zu einem DSP auf einem rc-Modul* erfordert ebenfalls den Eintrag der Boardkennung des Zielboards und zusätzlich der DSP-Kennung des Ziel-DSPs, so daß der PreSpike auf dem rc-Modul an den richtigen DSP übertragen wird.

4. Die Übertragung *vom lokalen DSP zum Host* erfolgt durch Setzen des Hostbits. Anson-

sten werden die lokalen Kennungen eingesetzt, da die protokollierende Sun Workstation über die Herkunft des PreSpikes informiert sein muß.

5. Die Übertragung *vom Host zum WMC des nrc-Moduls* erfordert den Eintrag der Zielkennungen. Die Neuronenadresse solcher externen PreSpikes wird in einem Adreßbereich oberhalb der lokalen Neuronenadressen gewählt.

Die CUs des nrc-Moduls transferieren grundsätzlich alle PreSpikes zur VME-Bus-Schnittstelle. Die dortige CU entscheidet über die Weitergabe an den WMC oder die Übertragung über den VME-Bus und korrigiert ggf. die Routinginformationen.

Auf einem *rc-Modul* muß auch die Übertragung der PreSpikes von einem lokalen DSP zu einem anderen lokalen DSP sowie von der VME-Bus-Schnittstelle zu einem lokalen DSP möglich sein, da die DSPs nur über den PreSpike-Austausch kommunizieren. Die Routing-Informationen sind daher auf den Ziel-DSP abgestimmt, eine Information über den Absender ist nicht enthalten. Der Ziel-DSP ordnet den empfangenen PreSpike als extern ein und adressiert damit einen eigenen lokalen Blockstartspeicher (ext-BSS) für insgesamt 16k Neuronen. Dieses Vorgehen führt dazu, daß sich Neuronen, die Ziele auf dem gleichen DSP haben, in ihrer lokalen Neuronenadresse auf ihrem Heimatprozessor unterscheiden müssen. Diese Unterscheidung muß vom MNET-Compiler (vergl. Kap. 5.2) bei der Vergabe der lokalen Neuronenadressen berücksichtigt werden. Um die kombinatorische Komplexität dieses Vergabeproblems zu mildern, werden die Neuronen für die rc-Module grundsätzlich lagenweise verteilt, was in Kap. 5.3 gute Ergebnisse gezeigt hat. Auf einem *rc-Modul* treten die folgenden Kommunikationsarten auf.

6. Die Übertragung *von einem lokalen DSP zu einem anderen lokalen DSP* erfolgt nur über das lokale CU-System, wobei der PreSpike mit der Board- und DSP-Kennung des Ziels versehen wird.

7. Die Übertragung *vom lokalen DSP zu einem DSP auf einem anderen rc-Modul* erfolgt ebenso unter Benutzung der VME-Bus-Schnittstelle. An dieser wird die Boardkennung des Zielboards beibehalten, so daß der PreSpike dem dortigen CU-System übergeben werden kann.

8. Die Übertragung *vom lokalen DSP zu einem nrc-Modul* erfolgt über die VME-Schnittstelle an den WMC des Zielboards. Der lokale DSP setzt seine eigene DSP-Kennung und die Boardkennung des Ziels, an der lokalen VME-Bus-Schnittstelle wird die Boardkennung auf die lokale Kennung zurückgesetzt.

9. Die Übertragung *vom lokalen DSP an den Host* erfolgt wie beim rc-Modul (4.).

10. Die Übertragung *vom Host zum rc-Modul* erfordert das Einsetzen der Board- und DSP-Kennung des Ziels. Bei der Vergabe der Neuronenadresse ist wiederum auf den Adreßbereich zu achten.

Die scheinbar große Vielfalt von zehn benötigten Kommunikationsarten läßt sich auf wenige Operationen herunterbrechen, die lokal an den betroffenen Komponenten durchgeführt werden. Im Konzept sind für die PreSpikes 14 bit als lokale Neuronenadresse (16k), 5 bit als DSP-Kennung (bis zu 32 DSPs) und 3 bit als Boardkennung (8 Module) vorgesehen, so daß insgesamt vier Millionen Neuronen adressiert werden können. Zu Details der Übertragungsmodi sei auf [Rah99][Rah00] zu den DSP-seitigen Kommunikationskomponenten, auf [Joc00] zur Übertragung über das CU-System, auf [Str00] zur Behandlung an der VME-Bus-Schnittstelle und auf [Ull00] zur Verarbeitung der PreSpikes durch den WMC verwiesen.

Die CU ist modular aus einem Kommunikationskern und mehreren Schnittstellenmodulen aufgebaut (Abb. 6-5). Aus diesen Modulen werden die benötigten CUs zusammengesetzt. Mehrere CUs werden dann in jeweils einem FPGA integriert.

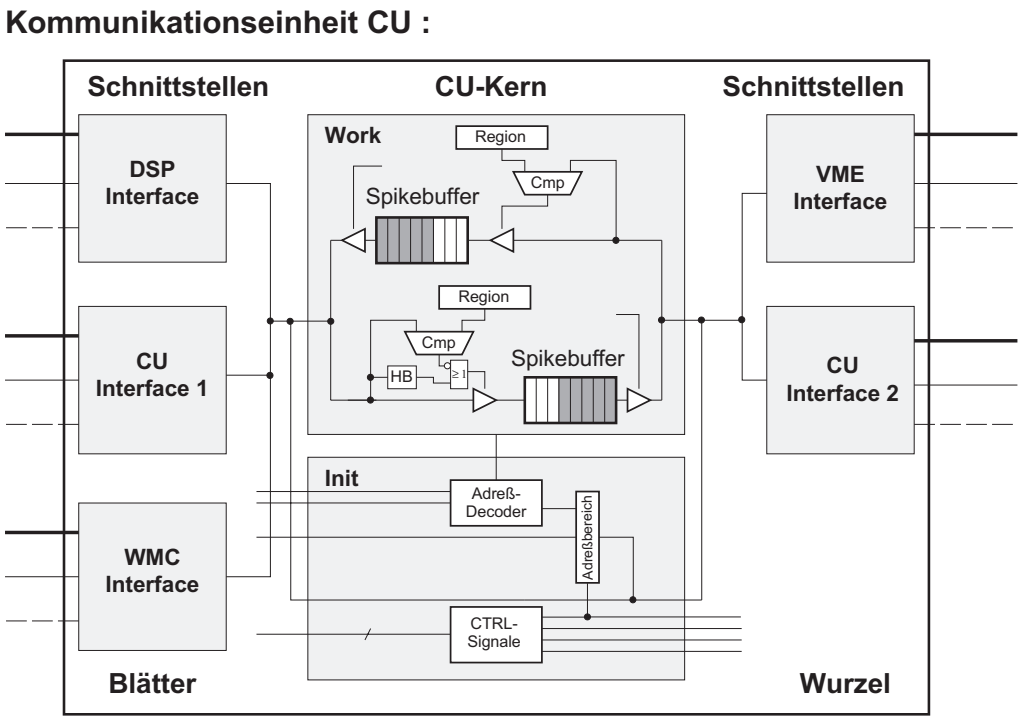

Abb. 6-5 Modularer Aufbau der Kommunikationseinheit CU [Joc00]

Der *CU-Kern* unterteilt sich wiederum in zwei Blöcke:

- Der *Work-Kern* ist in der Betriebsphase des ParSPIKE aktiv, d.h. während der Simulation der Zeitschritte. Er besteht aus zwei Spikepuffern für die beiden Übertragungsrichtungen der CU. Empfängt die CU von einer der angeschlossenen Schnittstellen Daten, d.h. vom Blätter-seitigen Bus oder vom Wurzel-seitigen Bus (siehe Abb. 6-2), so wird durch einen Vergleich der Zielkennung des empfangenen PreSpikes festgestellt, ob die CU dieses Datenwort auf den jeweils anderen Bus weiterschalten muß. Ist dieses der Fall, wird der PreSpike in den entsprechenden Spikepuffer eingetragen. Da für die Puf-

fer spezielle, in der verwendeten FPGA-Technik verfügbare 16 bit tiefe und 1 bit breite FIFO-Elemente benutzt wurden, sind die Puffer 16 x 32 bit tief ausgelegt. Die CU kann gleichzeitig mit der Blatt- und der Wurzelseite kommunizieren, wobei die Daten an die jeweiligen Schnittstellenmodule übergeben bzw. von diesen empfangen werden. Bei den CUs des nrc-Moduls ist die Kommunikation von der Wurzel zum Blatt nicht aktiv.

- Der *Init-Kern* ist während der Initialisierungsphase aktiv und schaltet die VME-Bus-Zugriffe der Sun Workstation zu den DSPs bzw. zum WMC durch. Über einen Adreß-Decoder wird festgestellt, ob die CU zu lesende Daten zurückgeben muß oder ob Zugriffe auf interne Register der CU erfolgen.

Die entworfenen FPGA-Designs für die CUs [Joc00] sehen einen internen Takt von 20 MHz vor, wobei über die Schnittstellenmodule mit asynchronen Protokollen kommuniziert wird.

Die Schnittstellenblöcke dienen zum Anschluß der CUs an die DSPs, den WMC sowie die VME-Bus-Schnittstelle und zur Kommunikation zwischen den CUs.

- Das *DSP-Interface* bedient den Multiprozessorbus der SHARC-DSPs. Über diesen Bus sind je 4 DSPs mit der CU verbunden. Zum Anschluß anderer Rechenknoten ist prinzipiell nur dieser Block des CU-Systems auszutauschen. Detaillierte Beschreibungen des DSP-Interfaces sind in [AD96] sowie in [Ibe99][Joc00][Sil00] zu finden. Der Multiprozessorbus kann PreSpikes mit einer Datenrate von maximal 10 MByte/s zu den DSPs übertragen, da vor jedem Zugriff im Vektorinterruptregister des DSPs kontrolliert wird, ob der DSP Daten annehmen kann. Dieser sammelt jeweils 16 empfangene PreSpikes in einem Puffer und löst dann einen Interrupt zur Übertragung in die lokale Spikeliste aus. Während dieser Übertragung kann der DSP keine Daten empfangen. Die CU sendet jeweils 8 PreSpikes pro Buszugriff und gibt dann den Bus wieder frei. Die DSPs können dagegen mit bis zu 20 MByte/s auf die CU schreiben.

- Das *CU2-Interface* verbindet bis zu vier CUs mit dem *CU1-Interface* der nächsthöheren CU. Dazu wird ein 32 bit breiter Bus benutzt, dessen Übertragungsrate bei 20 MByte/s liegt. Der Zugriff wird über ein einfaches Busprotokoll geregelt. In der Initialisierungsphase wird der Bus als gemeinsamer Daten- und Adreßbus benutzt. Das Multiplexen der Adressen und Daten dient zur Reduktion der Leitungszahl auf der Platine [Joc00].

- Das *WMC-Interface* verbindet die CU an der Wurzel des CU-Systems mit dem WMC. Das Protokoll entspricht dem CU1-CU2-Protokoll, wobei der WMC der einzige Teilnehmer ist und zudem nur auf ihn geschrieben wird [Str00][Ull00].

- Ein weiterer umfangreicher Schnittstellenblock ist das *VME-Interface*, das die Wurzel des CU-Systems mit einem Cypress VME-Master-Chipsatz verbindet (VIC068A) [Str00]. Dieser Block enthält einen Empfangspuffer und acht Sendepuffer für bis zu sieben andere ParSPIKE-Module und einen Host. Vom CU-System empfangene PreSpikes

werden in die entsprechenden Zielpuffer einsortiert, ggf. wird ihre Zielkennung korrigiert. Für jeweils 8 PreSpikes akquiriert der VME-Chipsatz den VME-Bus und führt eine Blockübertragung durch. Sende- und Empfangspuffer sind 16 Einträge tief, so daß auch Blocktransfers empfangen werden können. Das VME-Interface setzt zudem die VME-Zugriffe der Initialisierungsphase auf das Protokoll des CU-Systems um. Zum Anschluß des CU-Systems an ein alternatives Bussystem kann dieser Block ausgetauscht werden. Der VME-Bus in der für den ParSPIKE vorgesehenen 32 bit breiten Variante kann wie auch der Cypress-Chipsatz theoretisch bis zu 40 MByte/s übertragen. Durch die vorgesehenen nur 8 Worte langen Blocktransfers kann dieser Wert nicht erreicht werden, so daß etwa 10 MByte/s möglich erscheinen. Zur genauen Abschätzung dieses Wertes sind u.a. auch Angaben über das Verhalten einer VME-Bus-Sun bei der Bus-Vergabe notwendig, die nicht vorlagen. Da der VIC068A die Daten zur Blockübertragung von der CU-VME-Schnittstelle abruft, ist die Übertragungsleistung des VME-Busses für die erreichbare Übertragungsrate bestimmend [Str00].

Neben der Datenübertragung wird vom VME-Interface der Wurzel-CU auch die *Phasensynchronisation* mit den anderen ParSPIKE-Modulen vorgenommen. Dazu wird in einem Phasenregister (Abb. 6-6) an einer Bitposition, die der lokalen Boardkennung entspricht, ein Bit gesetzt, wenn alle lokalen Komponenten die Phase abgeschlossen haben. Der Registerinhalt wird dann über den VME-Bus auf die Phasenregister der anderen Module geschrieben, die ihn dem eigenen Register durch eine ODER-Verknüpfung überlagern.

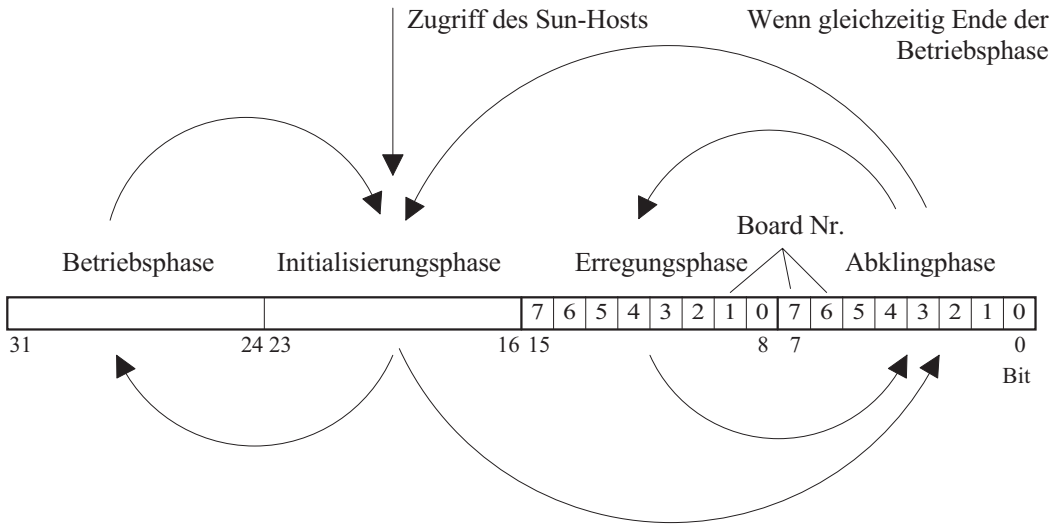

Abb. 6-6 Phasenregister zur Simulationsphasensynchronisation [Str00]

Sind die Bit für alle vorhandenen Module gesetzt, löscht das VME-Interface den lokalen Registerinhalt und schaltet in die nächste Phase. Durch ein Beschreiben des Registerabschnitts für die Initialisierungsphase zeigt das Sun-Hostsystem an, daß die Initialisierungsphase beendet ist, und stößt damit die Betriebsphase der Simulation an.

Die Realisierung des CU-Systems als hierarchisches Bussystem hat ihren Grund in der Einfachheit dieses Aufbaus und in der durch den gewählten DSP vorgegebenen Kommunikationsstruktur. Ein reines Bussystem stellt die einfachste und platzsparendste Realisierung der Kommunikation dar, ist jedoch aufgrund der Teilnehmerbeschränkung am SHARC-Multiprozessorbus nicht zu realisieren. Zudem nutzt ein solches reines Bussystem die Lokalität der Kommunikation nicht aus, so daß das gewählte hierarchische Bussystem den naheliegenden Kompromiß bildet.

Anhand der FPGA-Implementierungen in [Ibe99][Joc00][Str00] läßt sich die Übertragungsleistung für dieses System abschätzen. Vor dem Hintergrund, daß bei einer Zeitschrittlänge von 1 ms pro Rechenknoten 600 kByte/s pro Kommunikationsrichtung anfallen, erscheint das CU-System auch bei dem gewählten moderaten Takt von 20 MHz ausreichend dimensioniert. Erst bei den 32 Rechenknoten des rc-Moduls fallen auf einem Board überhaupt genug PreSpikes an, um das System auszulasten. Dabei ist zu beachten, daß ein PreSpike gewöhnlich nur an wenige Ziele gesendet wird und somit eine CU nur in einem ungünstigen Extremfall alle PreSpikes eines Moduls übertragen muß. Im Normalfall werden diese PreSpikes zu benachbarten DSPs übertragen, so daß ein Transfer über mehrere CUs oder über den VME-Bus nur für einen geringen Anteil erfolgt (siehe Kap. 7). Vor dem Hintergrund von im Mittel etwa 150 erzeugten PreSpikes pro Rechenknoten pro Zeitschritt (siehe Kap. 5.4) erscheinen auch die Puffertiefen von 16 Einträgen zum Abfangen von Schwankungen in der PreSpike-Menge ausreichend. Vor einem Phasenwechsel werden diese Puffer jeweils komplett geleert. Damit stellt das CU-System auch in seiner einfachen Struktur keinen Engpaß für die Simulation dar, so daß beim Einsatz genügend schneller Rechenknoten mit dem Erreichen der geforderten Simulationsleistung gerechnet werden kann. Durch die FPGA-Implementierung profitiert das CU-Design zudem vom Fortschritt in diesem Bereich.

6.4 Speichersubsystem

Neben dem CU-System wird aufgrund der speziellen Charakteristika der PCNN-Simulation auch für das Speichersubsystem des nrc-Moduls eine schaltungstechnische Realisierung benötigt. Dabei ist die Möglichkeit der Anbindung der Lerneinheit LU zu berücksichtigen, mit der einige der in [SH99][SSW00][Sch00] beschriebenen Lernverfahren in das ParSPIKE-System integriert werden können. Die Hauptkomponente des Speichersubsystementwurfs ist der *Weight Memory Controller* WMC, der in ein FPGA-Design überführt wurde [Ull00].

Das Speichersubsystem (siehe Abb. 6-7) realisiert den gemeinsamen Topologiespeicher und die gemeinsame Spikeliste in der nrc-Simulation. In der Abklingphase werden vom WMC die von den CUs übertragenen PreSpikes empfangen und in der Spikeliste abgelegt. Gleichzeitig benutzt die Lerneinheit (LU) den Topologiespeicher, um die Verbindungen durch Ler-

nen zu modifizieren. In der Erregungsphase liest der WMC die angelegte Spikeliste und sucht im Topologiespeicher die EIBs zu den PreSpikes. Die EIBs werden analysiert und die einzelnen EIB-Zeilen bzw. PostSpikes werden an die entsprechenden Ziel-DSPs versendet. Dazu wird je eine Linkverbindung pro DSP benutzt.

Neben der Spikeliste und dem EIB-Speicher wird auch ein Blockstartspeicher (BSS) benötigt, der Einträge für alle Neuronen im ParSPIKE-System enthält, da potentiell von allen diesen Neuronen PreSpikes empfangen werden können. Für das Lernen [SH99][SSW00] [Sch00] sind zudem einige weitere Speicherblöcke notwendig. Zum einen wird eine zweite Spikeliste benötigt. Die LU liest in der Abklingphase die PreSpikes des vorherigen Zeitschritts, da der WMC zeitgleich die neue Spikeliste schreibt. Weil die zu modifizierenden Verbindungen erst in der anschließenden Erregungsphase benötigt werden, kann das Lernen jeweils um einen Zeitschritt verzögert in der Abklingphase erfolgen. Da zum anderen für einige Lernverfahren Zugriff auf die Verbindungen der Vorgängerneuronen zu einem empfangenden Neuron erforderlich ist, wird neben dem senderorientierten EIB-Speicher ein empfängerorientierter inverser iEIB-Speicher benötigt. Dieser wird entsprechend über einen inversen Blockstartspeicher (iBSS) adressiert und enthält in den iEIBs neben der Neuronenadresse des Vorgängerneurons einen Zeiger auf die Verbindung im EIB-Speicher.

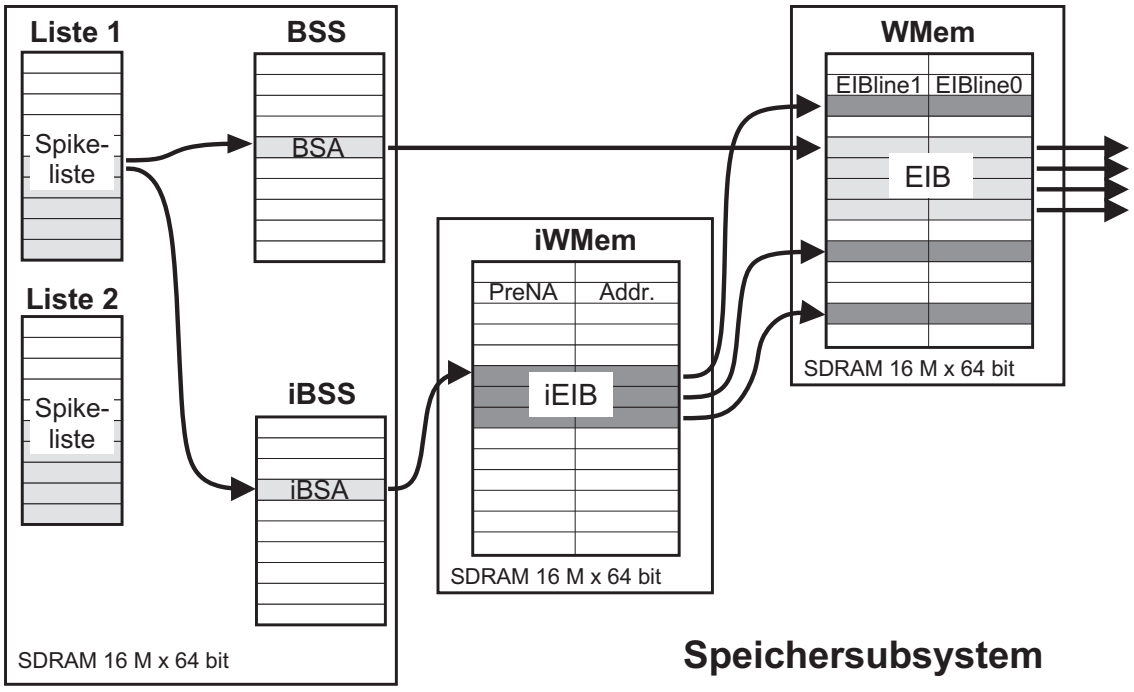

Abb. 6-7 Speicherblöcke des Speichersubsystems der nrc-Module

In dem in [Ull00] entworfenen Design des WMCs (siehe Abb. 6-8) und des Speichersubsystems werden alle benötigten Speicher in drei SDRAM-Riegeln (synchronous DRAM) mit je 128 MByte abgelegt. SDRAM verbindet die hohen Speicherdichten der DRAM-Technik

mit einem schnellen getakteten Zugriff. Die SDRAM-Riegel sind kompakt als 168-polige DIMM-Module (Dual Inline Memory Module) erhältlich. Eines dieser Module enthält die beiden Spikelisten, den BSS und den iBSS, sowie diverse Statusbits. Die beiden anderen Riegel dienen zur Aufnahme des inversen Topologiespeichers (iWMem) und des EIB-Speichers (WMem). Da die Speicher 64 bit breit angelegt sind, eine EIB-Zeile (EIB-Line) jedoch nur 32 Bit enthält, werden jeweils zwei EIB-Zeilen in einem Speicherwort abgelegt. SDRAM-Speicher erreichen ihre höchste Übertragungsrate beim sogenannten Burst-Zugriff, bei dem mehrere aufeinanderfolgende Datenworte im Block gelesen werden. Für den EIB-Speicher wird dieser Modus zum Lesen von je 4 Datenworten bzw. 8 EIB-Zeilen benutzt. Auf die Listen- und BSS-Speicher werden hingegen Einzelzugriffe ausgeführt, da ohnehin für jeden PreSpike bzw. jede Blockstartadresse (BSA) erst der EIB gelesen werden muß, bevor ein weiterer Zugriff erfolgt. Durch die Ablage der Listen und des BSS in einem eigenen Speichermodul kann das Lesen dieser Daten parallel zur EIB-Verarbeitung erfolgen. Die EIBs werden zunächst in einen Puffer der Tiefe 8 x 64 bit geschrieben. Von dort aus werden die EIB-Zeilen an die entsprechenden Linkports verteilt, die über 16 x 64 bit tiefe Puffer verfügen, wobei die Tiefe aus der verwendeten FPGA-Technik resultiert (siehe Kap. 6.3). Die Linkports lesen die Puffer und übertragen mittels eines Schieberegisters die EIB-Zeilen über die 4 bit breiten Linkverbindungen an die DSPs.

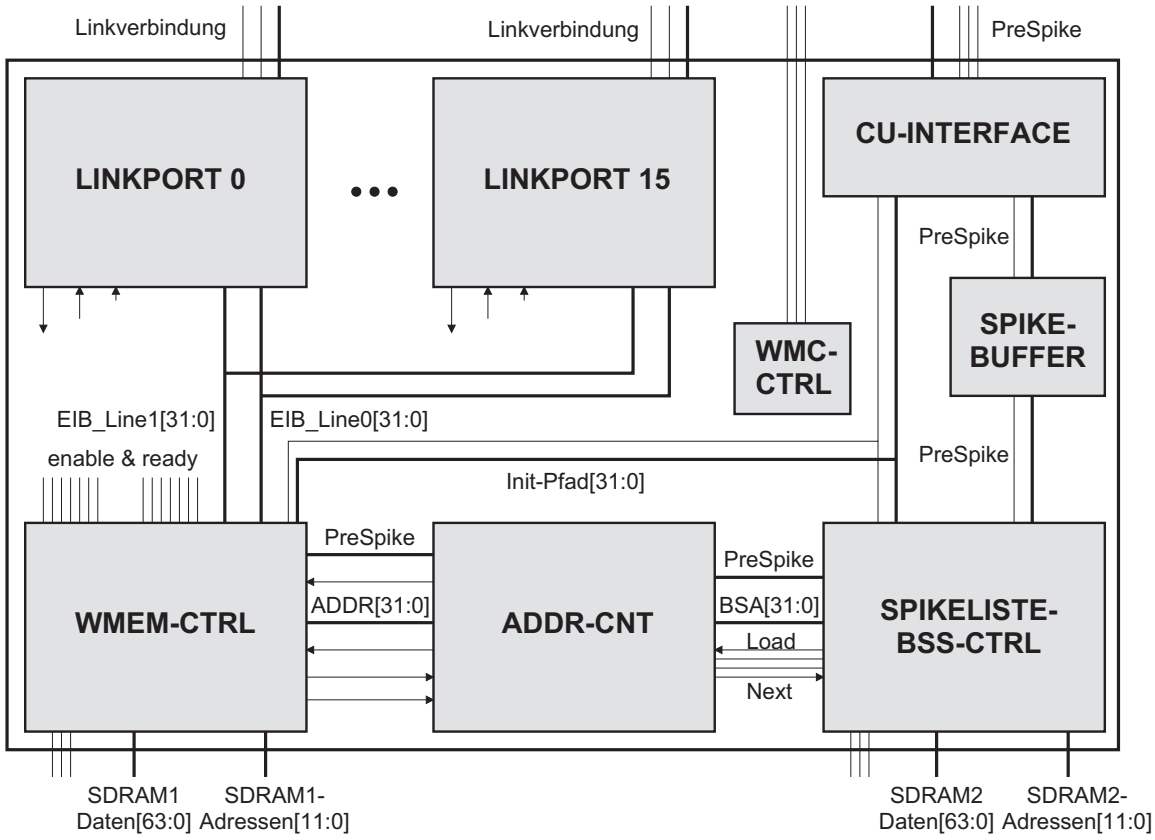

Abb. 6-8 Aufbau des Weight Memory Controllers WMC [Ull00]

Der WMC ist wie die CUs mit 20 MHz getaktet, wobei die SDRAM-Schnittstellen mit 40 MHz betrieben werden. Damit kann der WMC mit etwa 10 MByte/s PreSpikes in die Spikeliste schreiben und mit etwa 8 MByte/s PreSpikes mit den zugehörigen BSAs lesen [Ull00]. Der im Burstmodus betriebene EIB-Speicher (WMem) erlaubt das Lesen von EIBs mit einer Datenrate von 106 MByte/s. Die EIB-Zeilen können von den Linkports mit je 8 MByte/s an die DSPs übertragen werden, so daß die theoretische Übertragungsrate für 16 Linkports bei 128 MByte/s liegt. Gefordert war für die EIB-Übertragung jedoch eine Datenrate von 10 MByte/s pro Rechenknoten (siehe Kap. 5.4), die im aktuellen WMC-Design nicht erreicht wird. Allerdings ist bisher auch kein Prozessor verfügbar, der diese Datenrate tatsächlich abfordern würde. Der TigerSHARC, der die gewünschten Geschwindigkeiten annähernd erreicht, verfügt über 8 bit breite Linkverbindungen (gegenüber 4 bit beim ADSP21060), so daß durch eine leichte Modifikation der Linkports des WMCs eine ausreichende Übertragungsleistung erreicht wird. Da SDRAM-Speicher aktuell mit bis zu 133 MHz betrieben werden können, ermöglicht eine Erhöhung der bisher moderaten Taktfrequenz der SDRAM-Schnittstelle des WMCs auch hier ein Erreichen der gewünschten Werte. Neben diesen Gesamtdatenraten ist von Relevanz, wie die 16 EIB-Linkportpuffer durch die gelesenen Topologiedaten ausgelastet werden. Hier zeigt sich als Vorteil, daß der größte zusammenhängende Block von EIB-Zeilen jeweils den Heimatprozessor des verursachenden PreSpikes zum Ziel hat. Dieser Block muß nur selten übertragen werden, da er sich dort im EIB-Cache befindet. Der Rest des EIBs verteilt sich dann gleichmäßiger auf mehrere andere DSPs. Messungen mit dem PVM-Simulator für das Weitzelnetz ergaben für die Auslastungen der einzelnen Linkports Standardabweichungen von im Mittel +/- 16% bei einer Standardabweichung dieser Mittelwerte von +/- 5%. Damit ist die Auslastung sehr gleichmäßig, so daß die ermittelten Datenraten zur Abschätzung herangezogen werden können.

6.5 Realisierungsvorschläge

Das vorgestellte ParSPIKE-Konzept ist bisher nicht realisiert worden. Die Untersuchungen zum Konzept und der Vergleich mit anderen Konzepten und Realisierungen sollen vielmehr Aussagen über den Sinn einer Realisierung ermöglichen. Die im folgenden Kapitel vorgestellten Abschätzungen und Vergleiche dienen zur Einordnung des Konzepts, können aber nicht jedes Detail einer Realisierung vorhersagen. Insbesondere die Ermittlung der exakten zeitlichen Verläufe innerhalb der entworfenen Schaltungskomponenten erfordert Untersuchungen, die durch Simulationen nur mit unvertretbar hohem Aufwand durchzuführen sind. Dagegen steht die Flexibilität bezüglich Änderungen in der Prototypenphase einer Realisierung. Durch die Implementierung in FPGAs (Field Programmable Gate Arrays) lassen sich leicht Puffergrößen anpassen und Busprotokolle ändern, so daß eine Optimierung des Konzepts in der Prototypenphase erfolgt. Die vorgenommenen Untersuchungen und Abschät-

zungen geben dazu die groben Rahmenbedingungen an. Auf diesem Hintergrund wurden die Parameter für die entworfenen FPGA-Designs bewußt sehr konservativ gewählt, so daß Potential für die Ausnutzung der Möglichkeiten in diesem Bereich verbleibt. Dabei kommt einer Realisierung wiederum der technische Fortschritt im Bereich der FPGAs entgegen.

Die entworfenen FPGA-Designs [Joc00][Str00][Ull00] wurden mit entsprechenden Werkzeugen auf FPGAs der Bausteinfamilie Xilinx XC4000XV umgesetzt. Neben den speziellen Parametern dieser Bausteinreihe gibt Xilinx auch eine Anzahl von Gatteräquivalenten an, die eine Abschätzung der Größenordnung des Designs erlaubt. In Tabelle 6-2 sind die ermittelten Designgrößen für die entwickelten Bausteine angegeben. Es wurden vier unterschiedliche Designs entwickelt. Das *WMC-Design* implementiert den WMC eines nrc-Moduls in einem Baustein [Ull00]. Das *Blatt-CU-Design* implementiert jeweils zwei CUs mit DSP-Schnittstellen an den Blättern des CU-Baumes in einem Baustein und wird auf den rc- und den nrc-Modulen verwendet [Joc00]. Das *Wurzel-CU-Design des nrc-Moduls* enthält neben einer CU die VME- und WMC-Schnittstelle [Str00]. Das entsprechende *Wurzel-CU-Design des rc-Moduls* enthält insgesamt drei CUs und die VME-Schnittstelle [Str00].

Baustein	CLBs	I/Os	Takt	FPGA	Gatter
WMC	4.140	341	20/40 MHz	XC40150XV	150.000
Blatt-CU	588	264	20 MHz	XC4044XV	44.000
Wurzel-CU (nrc)	1.031	217	20 MHz	XC4044XV	44.000
Wurzel-CU (rc)	1.431	229	20 MHz	XC4052XV	52.000

Tab. 6-2 Parameter der FPGA-Designs für das ParSPIKE-Konzept

Neben der Xilinx4000-Bausteinserie existieren mittlerweile sowohl von der Firma Xilinx als auch von anderen Herstellern FPGA-Typen, die eine höhere Integration und außerdem höhere Takte erlauben. Die Codierung der Designs als VHDL-Beschreibungen [Joc00] [Str00][Ull00] ermöglicht prinzipiell die Umsetzung auf solche Bausteine. Ohnehin empfiehlt sich für eine Prototyprealisierung zunächst die Verwendung größerer Bausteine und die Bereitstellung zusätzlicher Verbindungen, um im Rahmen einer Inbetriebnahme die Flexibilität für Änderungen zu wahren. Zu beachten ist, daß durch die Nutzung einer speziellen Speicherstruktur der Xilinx-FPGAs für die jeweiligen Datenpuffer der CUs und des WMCs bei einem Wechsel zu einem anderen FPGA-Hersteller ggf. zusätzliche Ressourcen benötigt werden.

Mittels der entworfenen FPGAs ist der Aufbau von ParSPIKE-VME-Bus-Modulen möglich. Auf einem nrc-Modul sind 2 Blatt-CU-FPGAs, ein Wurzel-CU-FPGA (nrc) und ein WMC-FPGA mit 3 SDRAM-DIMM-Modulen zu integrieren. Dazu kommen die 16 DSPs und der VME-Bus-Chipsatz. Gegebenfalls ist hier ein weiteres FPGA und ein SRAM-Speicher zur Realisierung der Lerneinheit (LU) vorzusehen.

Auf einem rc-Modul sind 32 DSPs, dementsprechend 4 Blatt-CU-FPGAs und das Wurzel-CU-FPGA mit dem VME-Bus-Chipsatz unterzubringen. Diese VME-Bus-Platine erreicht die höchste Intergrationsdichte. Möglich wird die Unterbringung dieser hohen Bauteilanzahl durch eine beidseitige Bestückung (siehe Abb. 6-9 als Bestückungsbeispiel).

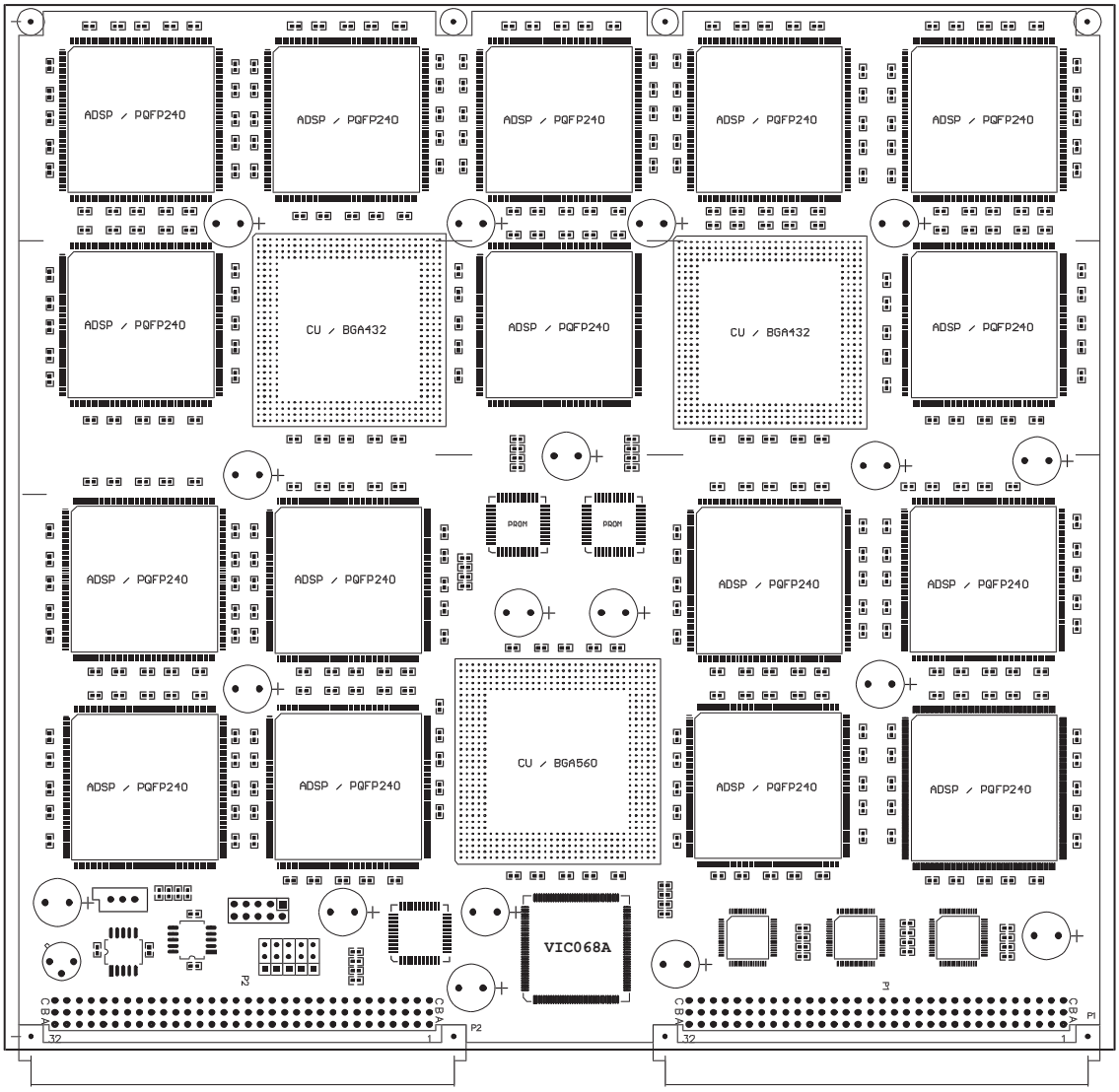

Abb. 6-9 Bestückungsbeispiel für den Aufbau eines rc-Moduls

Neben der Realisierung als VME-Bus-Module sind auch Module auf Basis des PCI-Busses denkbar, der als Erweiterungsschnittstelle in vielen PCs und Workstations vorhanden ist. Der Entwurf eines solchen Moduls hat gezeigt, daß auf einer PCI-Karte voller Baulänge in der rc-Variante bis zu 16 DSPs integrierbar sind (256k Neuronen) und in der nrc-Variante 8 DSPs (128k Neuronen) inklusive Speichersubsystem. Damit wird auf Basis eines handelsüblichen PCs mit 4 PCI-Modulen die Simulation von 1 Mio. Neuronen möglich.

Kapitel 7 - Einordnung und Evaluierung

Das vorgestellte ParSPIKE-Konzept ist im Rahmen der vorliegenden Arbeit nicht implementiert worden und damit auch keiner breiten Erprobung in der Praxis unterworfen worden. Dafür sind mehrere Gründe verantwortlich.

- Nach den Erfahrungen mit der Implementierung des SPIKE128k sollten vor der Implementierung eines Nachfolgesystems zuerst grundlegende Untersuchungen durchgeführt werden, die eine Aussage über das Potential einer parallelen Hardware auf Basis kommerzieller Prozessoren zulassen. Mit Hilfe der vorliegenden Ergebnisse kann bei einem zukünftigen Projekt eine Implementierungsentscheidung getroffen werden.
- Bisher liegen nur sehr wenige pulscodierte neuronale Netze in der relevanten Größenordnung vor. Es besteht daher erheblicher Entwicklungsaufwand, um überhaupt Sehsysteme auf PCNN-Basis zur Verfügung zu stellen, die dann in realen Szenarien unter Realzeitbedingungen evaluiert werden. Es erscheint daher sinnvoll, eine Implementierung eines Prototypsystems erst bei der Verfügbarkeit der Netze mit dann aktueller Technologie vorzunehmen.
- In verschiedenen Forschungsgruppen werden Projekte zur beschleunigten PCNN-Simulation bearbeitet, die nachfolgend mit dem vorgestellten Konzept verglichen werden. Erst auf Basis der Resultate dieser Arbeiten ist eine Entscheidung über die Implementierung des ParSPIKE-Systems sinnvoll.

Die nachfolgend dargestellten Leistungsabschätzungen beruhen auf Simulationen und Modellrechnungen. Simulationsdaten werden aus den Ergebnissen des in Kap. 5 vorgestellten PVM-Simulators gewonnen. Dazu werden die Szenarien des Weitzelnetzes und des Brausenetzes verwendet (siehe Kap. 2.4 und Kap. 5.3). Die Abschätzungen sind folglich nur exemplarisch, da aufgrund der wenigen verfügbaren Beispiele eine statistische Relevanz nicht gegeben ist. Wie nachfolgend gezeigt wird, entsprechen die Szenarien den Annahmen, die den Vergleichssystemen zugrundeliegen, so daß Tendenzen ablesbar sind.

Zur Überprüfung der Geschwindigkeit des gewählten Rechenknotens wurde eine Implementierung der Verfahren ausgeführt und mittels eines taktgenauen Simulatorprogramms mit realen Testdaten vermessen [Rah99][Rah00].

Um die physikalischen Kommunikationseigenschaften der DSPs zu überprüfen und Hinweise zur schaltungstechnischen Realisierbarkeit der ParSPIKE-Module zu erhalten, wurde ein Testsystem erstellt, das u.a. aus zwei DSPs und einem FPGA besteht [Ibe99][Sil00].

Aus den so gewonnenen Erkenntnissen und Daten können dann mit Hilfe der Ergebnisse der PVM-Simulationen Aussagen über die Simulationsgeschwindigkeit für die Beispielnetze auf dem ParSPIKE-System gemacht werden. Auf Basis dieser Daten wird der Vergleich mit anderen Simulatoren durchgeführt, wobei die Auswahl exemplarisch bleiben muß. Das Kapitel wird durch eine Wertung des Konzeptes unter Effizienzgesichtspunkten abgeschlossen.

7.1 DSP-Software-Testumgebung

Die Programmierung des ADSP21060 mit dem rc-Verfahren und dem nrc-Verfahren erfolgte mit Hilfe des EZ-Kit- und des VisualDSP-Entwicklungspakets von Analog Devices [AD96][AD00] im Rahmen der Studienarbeit [Rah99] und der Diplomarbeit [Rah00]. Die Pakete enthalten im wesentlichen einen C-Compiler, der eingebundene Assembler-Sequenzen verarbeitet, und einen DSP-Simulator, der taktgenau den Programmablauf auf dem DSP nachbildet und Verarbeitungszeiten mißt. Dem EZ-Kit liegt zudem ein Testboard mit einem Signalprozessor bei.

Mit Hilfe des C-Compilers wurden die C-Programme für die Workstation-Simulatoren auf den DSP umgesetzt. Da kein Betriebssystem eingesetzt wird, sind zuerst die Routinen für den Speicherzugriff und die Interrupt-Service-Routinen für die Kommunikation und Synchronisation in Assembler-Makros umgesetzt worden. Innerhalb der Speicherzugriffsfunktionen erfolgt aufgrund der fehlenden 16 bit Unterstützung zudem das Packen und Entpacken der Daten. Für jeden Neuronenparameter wurde eine eigene Funktion erstellt, da so mit festen Adreßoffsets gearbeitet werden kann und keine aufwendigen Adreßberechnungen anfallen. Das eigentliche Simulationsprogramm wurde schrittweise in seinen kritischen Programmteilen, d.h. den inneren Schleifen und häufig benötigten Funktionen, in Assembler-Makros umgesetzt. Die Verwendung von Makros ist notwendig, da der ADSP21060 über keine schaltungstechnische Unterstützung von Funktionsaufrufen verfügt und somit bei jedem Funktionsaufruf ein erheblicher Bedarf zur Sicherung von Registern wie dem Programmzähler anfällt. Durch diese Maßnahmen wurde die Ausführungsgeschwindigkeit des Programms vervierfacht.

Trotz der Nutzung von Assemblerbestandteilen wurde die modulare Programmstruktur der Workstation-Simulatoren soweit wie möglich beibehalten, um das Einfügen geänderter Verfahren in beiden Varianten zu ermöglichen. Außerdem war es durch den prinzipiell gleichen Programmaufbau und die ähnlichen Datenstrukturen möglich, zu bestimmten Zeitpunkten der Programmausführung mit der Workstation-Variante Testdaten für die DSP-Variante zu

erzeugen und die Ergebnisse der DSP-Variante dann mit denen der Workstation zu vergleichen. Abgesehen von leichten Rundungsunterschieden bei Fließkommaberechnungen [Rah99] stimmten die Ergebnisse überein.

Durch den modularen Aufbau konnten mittels des DSP-Simulators Ausführungszeiten für einzelne Programmabschnitte ermittelt werden (Tab. 7-1). Die Werte wurden in Taktzahlen gemessen und können durch Multiplikation mit der Taktdauer des Zielprozessors in Simulationszeiten umgerechnet werden. Die gemessenen Werte lassen sich den in Kap. 3.5 aufgelisteten statischen Parametern zur Ermittlung des Berechnungsaufwandes (AU*)* und des Kommunikationsaufwandes (KA) für ein Neuron zuordnen. Mit Hilfe der durch den PVM-Simulator ermittelten dynamischen Parameter kann der Gesamtaufwand berechnet werden.

Vorgang	lokal (nrc)	von extern (nrc)	rc-Variante
Abklingen A	220	-	220
Spikeerzeugung S	130	-	550
Blockstartspeicherzugriff B	180	-	270
Erregung E	65	150	100
Spikeempfang C	-	-	30

Tab. 7-1 Anzahl der Prozessortakte für einzelne Programmteile [Rah00]

Die Werte für die einzelnen Programmabschnitte müssen getrennt für die nrc-Programmvariante und die rc-Programmvariante betrachtet werden. Bei der nrc-Variante ist zudem zwischen lokalen Erregungen (E), die aus dem Cache bedient werden, und solchen, die extern vom WMC zugeführt werden zu unterscheiden. Die Spikeerzeugung (S) der rc-Variante dauert länger als bei der nrc-Variante, da über die Ziellisten die Ziel-DSPs ermittelt werden müssen. Beim Blockstartspeicherzugriff (B) der rc-Variante ist außerdem die Verarbeitung des Kopfes eines rc-EIBs eingerechnet. Der Aufwand zum Empfang und Eintrag eines PreSpikes in die Spikeliste (C) entfällt bei der nrc-Variante. Da bei der nrc-Variante im Mittel 40% der Erregungen aus dem Cache bedient werden (siehe Kap. 5.3), kann aus den beiden Werten für E ein gemeinsamer Wert von E = 120 errechnet werden. Alle Werte sind gerundete Richtwerte, die entsprechend der Beispielszenarien leicht differieren können.

Bei den gemessenen Werten für die Prozessortakte ist zu berücksichtigen, daß die bearbeiteten Daten jeweils erst von 32 bit auf 16 bit entpackt und nach der Bearbeitung gepackt werden müssen. Etwa 30% [Rah99] der Takte entfallen auf die Bearbeitung der entsprechenden Makros. Zu den Taktzahlen für die einzelnen Simulationsanteile tritt noch der Aufwand für das Rahmenprogramm, der aber vernachlässigbar gering ist. Auf Basis der ermittelten Werte lassen sich also Aussagen über die Programmlaufzeiten für verschiedene Szenarien machen, so daß die Simulationsgeschwindigkeit mit anderen Simulatoren verglichen werden kann.

7.2 Evaluierungshardware

Neben den Untersuchungen mit dem DSP-Simulator aus dem VisualDSP-Paket wurde eine eigene Entwicklungsumgebung implementiert und getestet. Dieses System (Abb. 7-1) ist in Form einer Platine mit zwei ADSP21060-Prozessoren, einem Xilinx XC4028 FPGA und einem SRAM-Speicher als EIB-Speicherersatz aufgebaut. Das FPGA wurde im Rahmen von [Ibe99] entwickelt, die Untersuchungen mit dem Testboard wurden in [Sil00] durchgeführt. Das System wird über einen Transputerrechner auf dem Testboard mit einer entsprechenden Transputerkarte in einer Sun Workstation verbunden.

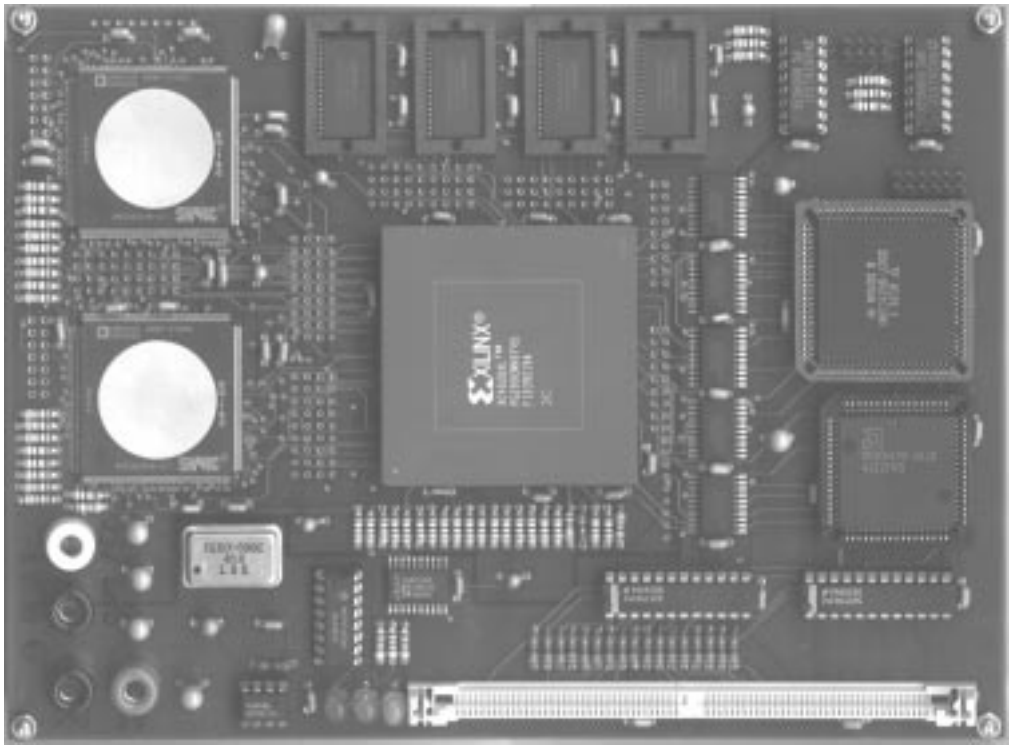

Abb. 7-1 Aufbau eines Testboards für die SHARC-DSPs [Sil00]

Im linken Bereich der Abbildung sind die zwei SHARC-DSPs zu sehen. In der Mitte des Boards befindet sich das Xilinx-FPGA und oberhalb dieses Bausteins ein 32 bit breiter SRAM-Speicher mit einer Kapazität von 512 kByte. Der rechte Teil des Boards wird (von oben nach unten) von der Transputer-Linkschnittstelle, einem T425-Transputer, einem Speicher-Controller auf Basis eines programmierbaren Bausteins und einem 4 MB PS2 Speichermodul eingenommen.

Das FPGA wird mit den Designs der Schnittstellenentwürfe für die CUs und den WMC geladen, deren Funktion und Geschwindigkeit dann über Testkontakte mit einem Logikanalysator ermittelt werden kann. Das FPGA kann im Betrieb vom Transputer aus mit neuen Designs geladen werden, die auf der Sun erzeugt werden. Der Transputer bietet den Vorteil,

daß er in einer höheren Programmiersprache (C) unter Rückgriff auf eine einfache Kommunikationsbibliothek programmiert werden kann, wobei die Kommunikationsschnittstelle auch auf der Sun zur Verfügung steht. Somit kann die Software sehr leicht im Rahmen der Tests angepaßt werden. Die Programmierung auf der Sun und dem Transputer sowie die Durchführung der Tests sind in [Sil00] beschrieben, die gemessenen Timing-Verläufe sind in die Aussagen bezüglich der Kommunikation mit den CUs und dem WMC eingegangen.

7.3 Leistungsabschätzung

Auf Basis der in Kap. 7.1 dargestellten Werte aus den DSP-Simulationen und der mit dem PVM-Simulator für zwei Beispiele ermittelten dynamischen Werte wird nachfolgend für zwei Szenarien beispielhaft die erreichbare Simulationsgeschwindigkeit des ParSPIKE-Systems berechnet.

Zum einen wird von einer Simulation des Weitzelnetzes auf dem nrc-Modul ausgegangen. Da dieses Netz mit ca. 120.000 Neuronen die Kapazität des Moduls von 256k Neuronen nicht auslastet, wird angenommen, daß zwei Weitzelnetze auf dem Modul simuliert werden. Damit entspricht die Neuronenverteilung auf die Rechenknoten der Verteilung bei der PVM-Simulation mit acht Workern, so daß die dort ermittelten Werte übernommen werden. Als statische Aufwandsparameter werden die Werte der nrc-Spalten aus Tab. 7-1 verwendet.

Mit den Werten für das Weitzelnetz (a = 2.204, s = 50, e = 4.047) ergibt sich damit nach (Gl. 3.25) der Simulationsaufwand AU (in Prozessortakten der SHARC-Baureihe).

$$\begin{aligned} AU &= a \cdot A + s \cdot S + s \cdot B + e \cdot E \\ &= 2204 \cdot 220 + 50 \cdot 130 + 50 \cdot 180 + 4047 \cdot 120 \\ &= 986020 \end{aligned} \qquad \text{(Gl. 7.1)}$$

Der maximal ausgelastete Rechenknoten bei der Verteilung des Weitzelnetzes auf 8 DSPs hat also 986.020 Takte zu rechnen. Dazu kommt ein geringer Anteil für das Rahmenprogramm, so daß im weiteren von etwa 1 Mio. Takten pro Zeitschritt ausgegangen wird.

Der Kommunikationsaufwand KA beschreibt die Anzahl der Speicherzugriffe während eines Zeitschritts. Dieser Aufwand ist in der gewählten Implementierung auf dem ADSP21060 irrelevant, da auf den internen Speicher genauso schnell zugegriffen wird, wie auf die Register des DSPs. Bei der Verwendung anderer Rechenknoten kann dieser Aufwand jedoch an Bedeutung gewinnen. Die Berechnung erfolgt nach (Gl. 3.26), wobei die Parameter b = 0,6 und c = 0,4 angenommen werden. Da 40% der EIB-Zugriffe aus dem Speicher des Rechenknoten bedient werden (Cachetreffer), ergibt sich EIB = 0,4. Für die Zugriffe auf die Neuronendaten wird N = 10 angenommen, alle anderen Parameter sind zu 1 gewählt.

Mit dem Kommunikationsaufwand der Hauptterme aus (Gl. 3.26) von

$$2N + (1 + b) \cdot AL = 21,6 \qquad \text{(Gl. 7.2)}$$

und

$$EIB + 2N + c \cdot AL = 20,8 \qquad \text{(Gl. 7.3)}$$

ergibt sich dann der Kommunikationsaufwand KA pro Rechenknoten (in Zugriffen).

$$\begin{aligned} KA &= 2204 \cdot 21,6 + 50 + 50 \cdot 2 + 4047 \cdot 20,8 \\ &= 131934 \end{aligned} \qquad \text{(Gl. 7.4)}$$

Im Falle der ParSPIKE-Module ist jedoch relevant, welche Datenmengen über das CU-System übertragen werden und welche Datenraten der WMC zu bewältigen hat. Die CUs haben dabei grundsätzlich alle PreSpikes zum WMC zu übertragen. Beim Weitzelnetz werden 404 PreSpikes pro Zeitschritt erzeugt, d.h. für zwei Weitzelnetze hat das CU-System etwa 808 PreSpikes pro ms zu übertragen, was einer Datenrate von 3,2 MByte/s entspricht. Zu beachten ist, daß nur in der Abklingphase, d.h. in der Hälfte des Zeitschritts, PreSpikes übertragen werden, so daß eine Datenrate von mindestens 6,4 MByte/s zur Verfügung stehen muß. Dieser Wert liegt wesentlich unter der für das CU-System und den WMC möglichen Datenrate von 20 MByte/s (siehe Kap. 6), so daß das PreSpike-Kommunikationssystem ausreichend dimensioniert ist. Die geforderte Datenrate für die PostSpike-Übertragung über die Linkports ergibt sich aus der mittleren Anzahl von Erregungen pro Rechenknoten (e = 3.516) abzüglich der mittleren Cachetrefferzahl (h_1 = 1.335), so daß im Mittel 2.181 PostSpikes pro Rechenknoten übertragen werden müssen, was einer Datenrate von etwa 8,8 MByte/s entspricht. Vor dem Hintergrund, daß nur in der Erregungsphase PostSpikes übertragen werden, sind etwa 20 MByte/s notwendig. Diese Übertragungsrate erlaubt der Linkport des ADSP21060, der WMC ist allerdings in der in [Ull00] durchgeführten Entwicklung nicht in der Lage, diese Datenrate zu liefern. Insbesondere erlaubt das SDRAM-Interface nicht das Lesen der benötigten 320 MByte/s, sondern erreicht nur 106 MByte/s. Zu bedenken ist jedoch, daß der ADSP21060 diese Datenraten nicht abfordern kann, da er eine Simulationszeit von 1 ms pro Zeitschritt nicht erreicht. Selbst für eine ParSPIKE-Implementierung auf Basis des TigerSHARC liefert daher der entworfene WMC die benötigten Datenraten.

Tatsächlich bilden also die entworfenen Schaltungskomponenten selbst in der wenig optimierten vorliegenden Version keinen Engpaß für die Simulationsgeschwindigkeit. Diese läßt sich aus der Anzahl von 1 Mio. Takten pro Zeitschritt und der Zykluszeit der einzelnen DSPs ermitteln. Entsprechend Tab. 6-1 ergibt sich für den ursprünglich ausgewählten ADSP21060 eine Zeit von 25 ms, für den Nachfolger ADSP21160 von 10 ms, für den TigerSHARC TS001 von 6,7 ms und für den TSxxx von 4 ms pro Zeitschritt. Zumindest die nrc-Simulationsleistung der TigerSHARC-Modelle liegt in der geforderten Größenordnung von 1 ms pro Zeitschritt, die technische Weiterentwicklung läßt die Erreichung des Wertes erwarten.

Für die Leistungsabschätzung des rc-Verfahrens werden die dynamischen Werte des Brausenetzes herangezogen. Dabei wird von einer Verteilung auf 24 Rechenknoten eines rc-Moduls ausgegangen, d.h. die in der PVM-Simulation für 12 Rechenknoten ermittelten Werte werden entsprechend reduziert. Es ergeben sich dann a = 11.264, s = 29, se = 453 und e = 15.451. Damit läßt sich wiederum der Rechenaufwand AU (in Takten) bestimmen.

$$\begin{aligned} AU &= a \cdot A + s \cdot S + (s + se) \cdot B + e \cdot E + se \cdot C \\ &= 11264 \cdot 220 + 29 \cdot 550 + 482 \cdot 270 + 15451 \cdot 100 + 453 \cdot 30 \\ &= 4182860 \end{aligned} \qquad \text{(Gl. 7.5)}$$

Bei der Simulation des Brausenetzes auf 24 DSPs muß also der einzelne DSP etwa 4,2 Mio. Rechentakte ausführen. Entsprechend ergeben sich für die verschiedenen Prozessoren Simulationszeiten pro Zeitschritt von 105 ms für den ADSP21060, 42 ms für den ADSP21160, 28 ms für den TS001 und 16,8 ms für den TSxxx.

Auch hier ist das Kommunikationsvolumen über das CU-Netz zu betrachten. Auf dem rc-Modul kommt dazu eine Säulenpartitionierung des Netzes zum Einsatz. Da von den 32 DSPs nur 24 benutzt werden, ist es denkbar, auf den verbleibenden 8 DSPs durch die Gangliendecoder Eingangsbilder in Spikes umrechnen zu lassen. Das CU-System hat dann insgesamt 453 PreSpikes pro Zeitschritt zu den DSPs und 696 PreSpikes von den DSPs zu anderen VME-Modulen zu übertragen. Damit ergibt sich eine Gesamtlast von etwa 2 MByte/s für die Übertragung zu den DSPs und 2,8 MByte für die Gegenrichtung. Insgesamt werden 4,8 MByte/s übertragen, wobei aufgrund der nur in der Abklingphase stattfindenden PreSpike-Übertragung eine Kapazität von 10 MByte/s zur Verfügung stehen muß. Diese Datenrate kann vom CU-System bewältigt werden. Auf die einzelnen DSPs muß nur mit 169 kByte/s geschrieben werden, der Rest des Systems kann bis zu 20 MByte/s übertragen. Auch im Fall des rc-Moduls stellt also die Rechenleistung des DSPs den limitierenden Faktor dar, während das CU-System genügend Kapazität aufweist.

Neben der Simulations- und Kommunikationslast auf den Modulen ist auch die Belastung des VME-Busses im Falle des Beispielszenarios (siehe Kap. 6.1) von Relevanz. Angenommen wird dabei, daß das Hostsystem die 453 PreSpikes pro Zeitschritt für das rc-Modul einspeist, dieses seine 696 PreSpikes pro Zeitschritt an eines der beiden nrc-Module überträgt und diese wiederum 25% ihrer insgesamt 1.616 Spikes als Ausgangssignale an den Host bzw. das andere Modul senden, d.h. 404 Spikes pro Zeitschritt. Insgesamt müssen also 1.553 PreSpikes pro Zeitschritt übertragen werden. Da die Übertragung wiederum nur in der Abklingphase erfolgt, muß eine Datenrate von etwa 12,5 MByte/s auf dem VME-Bus zur Verfügung stehen. Erst wenn die Prozessoren eine Simulation eines Zeitschrittes in 1 ms zulassen, wird der VME-Bus mit einer solchen Datenrate ausgelastet.

Nachdem nunmehr für die Beispielszenarien die Berechnung der wesentlichen Simulationszeit- und Kommunikationsparameter gezeigt wurde, sollen nachfolgend die wichtigsten Da-

ten abhängig von dem verwendeten Rechenknoten zusammengefaßt werden. Die Datenraten für die CUs, den WMC und den VME-Bus sind dabei an die von den Rechenknoten erzeugten Datenraten angepaßt, d.h. entsprechend der Rechenzeit reduziert. Dargestellt ist ein Szenario für eine ParSPIKE-Realisierung mit *vier nrc-Modulen*, die acht Weitzelnetze mit zusammen 915.208 Neuronen und 30.112.960 Verbindungen simuliert (Tab. 7-2), ein System mit *zwei rc-Modulen*, das zwei Brausenetze mit zusammen 736.372 Neuronen und 41.747.970 Verbindungen in Form von rc-EIBs berechnet (Tab. 7-3), sowie der *Mischaufbau aus zwei nrc- und einem rc-Modul*, mit dem vier Weitzelnetze und ein Brausenetz mit zusammen 825.790 Neuronen und 35.930.465 Verbindungen simuliert werden (Tab. 7-4).

Prozessor	AD21060	AD21160	TS001	TSxxx
Zeit pro Zeitschritt in ms	25	10	6,7	4
CU-Datenrate (nrc) in MByte/s	0,26	0,64	0,96	1,6
WMC-Datenrate in MByte/s	12,8	32	47,8	80
Link-Datenrate in MByte/s	0,8	2	3	5
VME-Datenrate in MByte/s	0,2	0,5	0,75	1,25

Tab. 7-2 Geschwindigkeiten und Datenraten für vier nrc-Module

Prozessor	AD21060	AD21160	TS001	TSxxx
Zeit pro Zeitschritt in ms	105	42	28	16,8
CU-Datenrate (rc) in MByte/s	0,1	0,26	0,39	0,65
VME-Datenrate in MByte/s	0,2	0,52	0,78	1,3

Tab. 7-3 Geschwindigkeiten und Datenraten für zwei rc-Module

Prozessor	AD21060	AD21160	TS001	TSxxx
Zeit pro Zeitschritt in ms	105	42	28	16,8
CU-Datenrate (rc) in MByte/s	0,1	0,26	0,39	0,65
CU-Datenrate (nrc) in MByte/s	0,06	0,15	0,23	0,38
WMC-Datenrate in MByte/s	3,05	7,62	11,43	19,05
Link-Datenrate in MByte/s	0,19	0,48	0,71	1,19
VME-Datenrate in MByte/s	0,12	0,3	0,45	0,74

Tab. 7-4 Geschwindigkeiten und Datenraten: ein rc- und zwei nrc-Module

Insgesamt zeigt sich, daß das ParSPIKE-Konzept auf Basis der aktuellen technischen Möglichkeiten die Simulation von Netzen mit 1 Mio. Neuronen in 4 ms zuläßt, wenn das Weitzelnetzszenario zugrunde gelegt wird, und im Brausenetzszenario immerhin Geschwindigkeiten bis zu 16,8 ms pro Zeitschritt erlaubt. Dazu muß jedoch der angekündigte TSxxx mit einer Taktfrequenz von 250 MHz vorausgesetzt werden. Die Übertragungsmöglichkeiten der entwickelten Schaltungskomponenten werden erst von diesem Prozessor benötigt, für die anderen SHARC-Prozessoren sind die Komponenten überdimensioniert. Im Rahmen der weiteren Entwicklung der Prozessortechnik sind zuerst für den WMC, der den gemeinsamen Topologiespeicher der nrc-Module kontrolliert, Engpässe zu erwarten, die jedoch im Rahmen der gleichzeitig fortschreitenden FPGA-Technik auflösbar sind. Bei der Erreichung der Zielgeschwindigkeit von einem Zeitschritt pro Millisekunde (ms) ist auch der VME-Bus ausgereizt, so daß ggf. der Rückgriff auf den schnelleren PCI-Bus sinnvoll erscheint. Insgesamt ist in der Perspektive das Ziel eines parallelen Systems auf Basis kommerzieller Mikroprozessoren mit einer Simulation von 1 Mio. Neuronen in 1 ms pro Zeitschritt erreichbar.

7.4 Vergleichssysteme

Um die Effizienz der vorgeschlagenen Lösung beurteilen zu können, soll ein Vergleich mit anderen verfügbaren oder konzeptionierten Simulatoren erfolgen. Die Auswahl erfolgt exemplarisch, wobei entsprechend verschiedener Klassen von alternativen Simulatorsystemen Beispiele ausgewählt wurden. Zunächst einmal werden Alternativen zu dem bezüglich seiner Rechenleistung bisher unterdimensionierten SHARC-DSP als Rechenknoten für das ParSPIKE-System vorgestellt. Daran schließt sich die Beschreibung einiger digitaler Neuroprozessoren an, die speziell für die PCNN-Simulation entwickelt wurden. Zudem werden die darauf basierenden Systeme erläutert. Weiterhin wird die Möglichkeit der PCNN-Simulation auf kommerziell erhältlichen Neurocomputern erwogen. Eine mögliche Alternative zu den ParSPIKE-Modulen in Form von anderen DSP-Parallelrechnern wird geschildert. Die Aufstellung schließt mit den Softwaresimulatoren, wobei insbesondere Varianten für parallele Rechensysteme von Interesse sind. Soweit möglich, werden die Leistungsdaten der Alternativsysteme mit den ParSPIKE-Daten verglichen.

7.4.1 Alternative Rechenknoten

Zu Beginn der ParSPIKE-Konzeption wurden sämtliche alternativen Rechenknoten verworfen, da sie über keinen ausreichenden on-Chip-Speicher verfügten bzw. bei der Verwendung von externem Speicher in bezug auf den Platzverbrauch und damit die erreichbare Modulleistung nicht mit dem SHARC konkurrieren konnten [JSR97][Rah99][WHR99b]. Aktuell hat sich dieser Zustand durch das Erscheinen weiterer Hochleistungssignalprozessoren und durch den Trend zur Integration größerer Speicher im Microcontroller-Bereich geändert.

Insbesondere in Form der *Texas Instruments* TMS320C6x-Baureihe ist eine interessante Alternative zu den SHARC entstanden [TI00]. Der TMS320C6203 verfügt über bis zu 896 kByte on-Chip-Speicher und ist mit einer Taktfrequenz von 300 MHz angekündigt. Damit dürfte der TMS dem TigerSHARC in der Simulationsleistung überlegen sein, wobei der gleiche hohe Integrationsgrad erreicht wird. Allerdings verfügt er über keine Multiprozessorunterstützung und insbesondere keine Linkschnittstellen, so daß ein geändertes CU-System vonnöten wäre. Für die TMS320C6x-Reihe sind von Texas Instruments Taktfrequenzen von bis zu 1,1 GHz angekündigt. Damit ist die gewünschte Simulationsleistung erreichbar.

Die Firma *Lucent Technologies* positioniert die DSP16000-Reihe im Bereich des Hochleistungsrechnens [Lu00]. In den aktuellen Ausführungen verfügt der DSP16410 über einen on-Chip-Speicher von 388 kByte, der prinzipiell die Integration der ParSPIKE-Verfahren möglich erscheinen läßt. Mit aktuell 150 MHz Taktfrequenz erscheint auch hier die Perspektive zur Erreichung der benötigten Simulationsleistung gegeben. Eine Multiprozessorunterstützung und Linkports sind wie beim TMS nicht vorhanden. Interessant ist der Prozessor aufgrund des gegenüber dem TigerSHARC halbierten Platzbedarfs (17 x 17 mm).

Neben den DSPs werden durch die Integration von embedded DRAM zunehmend auch Microcontroller als Rechenknoten interessant. In diesem Bereich verfügt der *Mitsubishi* M32R/D über einen 2 MByte großen on-Chip-Speicher [MS00]. Dieser Prozessor ist in einem nur 16 x 16 mm großen Gehäuse erhältlich. Er verfügt jedoch mit einer Taktfrequenz von 66 MHz nicht über die benötigte Rechenleistung und zudem nur über rudimentäre Kommunikationsschnittstellen. Der sehr große Speicher eröffnet allerdings neue Perspektiven für die Simulationsverfahren, z.B. in Bezug auf die Topologiespeicherung. Zudem ermöglicht auch der Energiebedarf von 750 mW eine erhöhte Anzahl von Rechenknoten pro Modul.

Insgesamt zeigt sich, daß die Entwicklung im kommerziellen Prozessorbereich dem ParSPIKE-Konzept in großen Schritten entgegenkommt. Dieses war eine der Absichten bei der Entwicklung eines Systems auf Basis von Standardkomponenten.

7.4.2 SPIKE128k

Der SPIKE128k [FBH95][FH95][FHJ99][Fra97][HFS97] (siehe auch Kap. 3.5) beruht im Gegensatz zum ParSPIKE auf der Entwicklung einer speziellen Prozessoreinheit für die PCNN-Simulation. Trotz der Tatsache, daß es sich um einen Einzelprozessor mit nur 10 MHz Grundtakt handelt, kann das System ein einzelnes Weitzelnetz mit 114.401 Neuronen in 11 ms pro Zeitschritt simulieren und erreicht damit die Leistung von 8 ADSP21160-Prozessoren. Gerade der sehr hohe Entwicklungsaufwand und die geringe Flexibilität dieser Spezialhardware waren jedoch der Grund für die ParSPIKE-Entwicklung. Trotzdem zeigt der SPIKE128k, welches Potential in Spezialprozessoren für die PCNN-Simulation liegt, so daß diese Möglichkeit eine deutliche Konkurrenz zum ParSPIKE bildet.

7.4.3 NESPINN und MASPINN

Am Institut für Mikroelektronik an der TU Berlin wurden in der Arbeitsgruppe von Prof. Klar Konzepte zur mikroelektronischen Integration von Spezialprozessoren für die PCNN-Simulation entwickelt und z.T. realisiert [SAM00]. Diese Arbeiten standen im Zusammenhang mit der SPIKE128k-Entwicklung und verwenden teilweise die gleichen Konzepte. Im gleichen Zusammenhang wurden auch Untersuchungen zur PCNN-Simulation auf alternativen Plattformen angestellt [JRK95][JRS98][JSR97][RJK95][SJR98].

Der NESPINN-Prozessor [JRK96][REJ97][RJK97] wurde mit einer Taktfrequenz von 50 MHz ausgelegt und verarbeitet dendritische Potentiale mit vier Prozessorpipelines. Über eine Markierung wird eine der Abklingliste ähnliche Beschränkung auf aktive Potentiale erreicht. Die Neuronendaten sind in externen SRAM-Speichern abgelegt, für die Topologieverarbeitung und die Spikelistenverwaltung sind ebenfalls externe Einheiten vorgesehen.

Der MASPINN-Prozessor [SMJ98][SMK98] nimmt das NESPINN-Konzept auf und optimiert insbesondere die Nutzung des externen SRAM-Speichers, um seine vier Pipelines möglichst kontinuierlich mit Daten zu speisen. Durch eine Vorverarbeitung der Topologiedaten in Form einer Akkumulation der Potentialmodifikationen durch die Verbindungsgewichte mittels einer externen Topologiedateneinheit wird eine Verschränkung der Abkling- und Erregungsphase erreicht, wobei die längere Phase die Simulationszeit bestimmt. Der MASPINN-Prozessor wurde realisiert und befindet sich aktuell in der Testphase [SAM00]. Da die externe Topologieeinheit bisher nicht realisiert wurde, können Geschwindigkeitsschätzungen nur für die MASPINN-Berechnungen durchgeführt werden. Die Taktfrequenz des Prozessors beträgt 100 MHz.

Da die Geschwindigkeitsschätzungen für die MASPINN- und NESPINN-Prozessoren nicht auf dem Weitzelnetz, sondern auf einem in [SRE96] beschriebenen Netz mit ähnlicher Charakteristik beruhen, sind die in Tabelle 7-5 gegenübergestellten Daten für das ParSPIKE-Konzept und den SPIKE128k (ggf. mehrere gekoppelte Systeme) entsprechend modifiziert.

Anzahl der Neuronen	128k	512k	1 M
MASPINN (4 PE/100 MHz)	0,8 ms	3,2 ms	6,4 ms
NESPINN (4 PE/50 MHz)	3 ms	11 ms	23,4 ms
SPIKE128k (1-8 PE/10 MHz)	10 ms	11 ms (4PE)	12 ms (8PE)
ParSPIKE (64 PE/AD21060/40 MHz)	2,5 ms	10 ms	20 ms
ParSPIKE (64 PE/AD21160/100 MHz)	1 ms	4 ms	8 ms
ParSPIKE (64 PE/TS001/150 MHz)	0,7 ms	2,7 ms	5,4 ms
ParSPIKE (64 PE/TSxxx/250 MHz)	0,4 ms	1,6 ms	3,2 ms

Tab. 7-5 Vergleich der Simulationsgeschwindigkeiten [SMJ98][SAM00]

7.4.4 Sonstige Spezialhardware

Neben dem SPIKE128k und den NESPINN/MASPINN-Prozessoren ist eine Vielzahl von Hardwareimplementierungen für PCNN vorgenommen worden [MB98]. Interessant ist eine an der Universität Dortmund entstandene Implementierung in Form eines FPGA-Prozessors, die für immerhin 32k Neuronen auf Basis eines Systems von acht VME-Modulen die echtzeitnahe Simulation von PCNN erlauben soll [RHG96]. Das *Endeavour*-Projekt verbindet die Flexibilität einer FPGA-basierten Lösung mit der Geschwindigkeit eines Spezialprozessors. Die mögliche Netzgröße erreicht jedoch nicht die Größenordnung einer ASIC-Lösung wie MASPINN und liegt daher nicht in einem für diese Arbeit relevanten Rahmen.

Eine weitere interessante Implementierung stellt der *Silicon Cortex SCX* dar [DDW98], der eine PreSpike-basierte Kommunikation auf Basis von Neuronenadressen bietet. Dabei wird in Form eines *Address Event Bus* auf Ebene eines VME-Moduls ein Broadcast-Bus bereitgestellt. Auf den angeschlossenen Recheneinheiten wird für die dort simulierten Neuronen mittels einer empfängerorientierten Topologiespeicherung ermittelt, ob der PreSpike zu Erregungen führt. Der Silicon Cortex erlaubt die Integration schneller analoger Neurochips. Allerdings ist das System aufgrund der Nachteile des Broadcast-Busses auf einige tausend Neuronen pro VME-Modul begrenzt.

Insgesamt existiert abgesehen von der NESPINN/MASPINN-Architektur aktuell keine PCNN-Spezialhardware, die Netzgrößen bis zu 1 Mio. Neuronen unterstützt.

7.4.5 Neurocomputer

Neben der Entwicklung von Spezialhardware für die PCNN-Simulation existiert eine breite Palette von Systemen für die Simulation konventioneller neuronaler Netze [Zel94][KS96]. Diese Systeme können z.T. auch für die PCNN-Simulation eingesetzt werden. Dabei sind Implementierungen auszunehmen, die sehr speziell auf bestimmte Eigenschaften der zu simulierenden Netze eingehen, wie z.B. das bekannte SYNAPSE-System [Ram92], das insbesondere Matrix-Vektor-Berechnungen beschleunigt. Solche Systeme sind für die deutlich anderen Anforderungen der PCNN-Simulation zumeist ineffizient. Simulatoren, die auf programmierbaren Prozessoren beruhen, können hingegen an die PCNN-Erfordernisse angepaßt werden. Auf dem CNAPS-Neurocomputer sind PCNN-Implementierungen durchgeführt worden [JRK95][JSR97][KL99], die bei Netzen bis zu einigen zehntausend Neuronen gute Ergebnisse erzielen. Dabei werden sehr reguläre Verknüpfungsstrukturen vorausgesetzt. Bei größeren Netzen sind die Grenzen des Systems schnell erreicht. Für ein Netz mit 16k Neuronen ist in eine Simulationszeit von 1,5 ms pro Zeitschritt ermittelt worden, bei 128k Neuronen reicht die Simulationsleistung mit ca. 1 s pro Zeitschritt nicht mehr aus [JRK95]. Konventionelle Neurocomputer sind insgesamt aufgrund der Optimierung auf die Anforderungen einfacher Neuronenmodelle für die PCNN-Simulation wenig geeignet.

7.4.6 DSP-Parallelrechner

Interessante Alternativen zum ParSPIKE-Konzept sind sicherlich auch im Bereich der DSP-Parallelrechner zu suchen. Solche Systeme sind in diversen Projekten als Plattform für parallele Simulationen neuronaler Netze eingesetzt worden.

Im Bereich der PCNN-Simulation wurde ein DSP-Parallelrechnersystem positioniert, das an der Universität Duisburg und am Fraunhofer-Institut in der Arbeitsgruppe von Prof. Hostikka entwickelt wurde [RHH90][Sch93][SHK94]. In diesem System findet eine Simulation mit verteilter Netztopologie statt, wobei die Kommunikation mit Hilfe von PostSpikes abgewickelt wird. Diese Art der Kommunikation wurde in Kap. 4.2 verworfen, da das Aufkommen bis zu zwei Größenordnungen über dem PreSpike-Aufkommen liegt. Das System basiert auf einem Texas Instruments TMS320C40 und einer speziellen Kommunikationshardware. Bezüglich Rechenleistung und lokalem Speicher ist das System nur für einige tausend Neuronen ausreichend. Zur Partitionierung der Netze wurden die in [Kre93][KSE92] [KSS93] vorgeschlagenen schnittoptimierenden Ansätze favorisiert (siehe auch Kap. 4.4), die im Rahmen der vorliegenden Arbeit zu keinen befriedigenden Ergebnissen führten.

Ein weiteres DSP-System (MUSIC) wurde in [MGG95] vorgestellt und als Plattform für die Simulation konventioneller neuronaler Netze positioniert. Es basiert auf dem Motorola DSP 96002 und bis zu 3MB großen lokalen Speichern. Die Kommunikation wird über einen einzelnen mit 5 MHz getakteten Broadcastbus abgewickelt, an den die Rechenknoten über Xilinx-FPGAs angeschlossen sind. Aufgrund des gegenüber der PCNN-Verarbeitung unterschiedlichen Simulationsparadigmas und des unterschiedlichen Kommunikationsprinzips ist dieses System mit dem ParSPIKE-Konzept kaum vergleichbar.

Neben speziellen Systemen für die Neuronale-Netze-Simulation existiert eine Vielzahl von DSP-Parallelrechnern. Prinzipiell kommen diese kompakten Systeme mit großer Rechenleistung auch für die PCNN-Simulation in Frage. Insbesondere auf Basis des SHARC sind von verschiedenen Anbietern VME-Platinen mit bis zu 12 DSPs im Angebot [AD00]. Allen diesen Systemen ist jedoch gemein, daß sie Betriebssysteme auf dem Rechenknoten einsetzen und über Software-Protokolle kommunizieren. Damit sind weder die Speicherplatzanforderungen noch die Kommunikationsleistungen für die PCNN-Simulation zu befriedigen. Ein DSP-Parallelrechner auf Basis der SHARC-Prozessoren mit spezieller Kommunikationshardware wurde an der TU Hamburg-Harburg entwickelt [May97][May98][May99]. Dieses System ist besser skalierbar als einfache SHARC-VME-Platinen (bis zu 512 Prozessoren), jedoch kann es ebenfalls die PreSpike-Kommunikation nicht in Hardware abwickeln. Damit ist die Simulationsleistung von konventionellen Parallelrechnern bei einer etwas höheren Integration zu erwarten, wobei jedoch eine Festlegung auf den SHARC als Rechenknoten erfolgt. Gerade die Forderung nach einer speziellen Kommunikationshardware führte zur Konzeption des ParSPIKE und zum Ausschluß kommerzieller DSP-Parallelrechner.

7.4.7 Software-Simulatoren

Wenn auf eine Echtzeitsimulation verzichtet werden kann, bietet sich aufgrund der überlegenen Flexibilität eine Simulatorimplementierung in Software auf einer Standard-Workstation oder einem kommerziellen Parallelrechner an.

Solche Simulationen wurden insbesondere für einfache Neuronenmodelle und spezielle Netztypen (z.B. Backpropagation) durchgeführt [KS96][Mis97][Zel94]. Da auch diese neuronalen Netze inhärent parallel aufgebaut sind, wurden viele Simulationen auf Parallelrechnersystemen durchgeführt. Ein Beispiel für diesen Ansatz bietet ein in der Arbeitsgruppe von Prof. Rammig an der Universität Paderborn durchgeführtes Projekt auf Parsytec Parallelrechnern [ABC92][ABC93][Res93a][Res93b][Res93c][Dit94][Str94][Res99]. Ausgehend von der Annahme, daß in konventionellen neuronalen Netzen ohnehin alle Neuronen in jedem Zeitschritt zu berechnen sind und mithin ihr Zustand auch über alle Verbindungen übertragen werden muß, wurde eine rein zeitgetriebene Simulation implementiert, die in jedem Zeitschritt alle Neuronen und alle Verbindungen berechnet. Die parallele Simulation erfolgt durch Verteilung der Neuronen und eine Netztopologiespeicherung auf dem Rechenknoten des sendenden Neurons, so daß die Kommunikation über postsynaptische Erregungen abgewickelt wird. Damit ist die Netzverteilung unabhängig vom Eingangsreiz, so daß sich das Verteilungsproblem auf ein Mapping des Netzgraphen auf einen Hardware-Graphen reduziert, mit der Maßgabe, die Anzahl der für eine Verbindung zu überwindenden Rechenknoten gering zu halten. Insbesondere [Res99] widmet sich Verfahren zur Graphenpartitionierung unter der genannten Randbedingung. Soweit Simulationsergebnisse verfügbar sind [Res93b][Dit94][Str94], fällt die Beschleunigung durch die Parallelisierung gering aus (unterhalb Faktor 4). Dieses Ergebnis kommt aufgrund des hohen Maßes an Kommunikation bei gleichzeitig geringem Rechenaufwand zustande. Gerade diese Bedingungen sind bei PCNN anders und führen dort zu einer besseren Parallelisierbarkeit.

Im Rahmen des NESPINN/MASPINN-Projekts an der TU-Berlin (siehe Kap. 7.4.3) wurden auch Untersuchungen mit einem sequentiellen Softwaresimulator *SimSPINN* durchgeführt [WWK97]. Mit diesem Simulator wurden einige Geschwindigkeitsmessungen auf Workstations und PCs durchgeführt, die zu ähnlichen Ergebnissen wie in Kap. 3.5 dargestellt kamen. Ein Netz mit 128k Neuronen wurde dabei in 85 ms pro Zeitschritt auf einem Pentium II PC mit 266 MHz simuliert. Zudem wurde eine der MNET-Sprache vergleichbare Netzbeschreibungssprache BNET vorgeschlagen, zu der jedoch kein Compiler existiert. Mit einem ähnlichen Softwaresimulator und einem nicht parallelisierenden MNET-Compiler [Möl95] (siehe auch Kap. 5.2) wurden auch die Untersuchungen an der Universität Marburg durchgeführt, bei denen auf einer DEC Alpha 250 mit 266 MHz Taktfrequenz eine Simulationszeit von 140 ms pro Zeitschritt für das Weitzelnetz gemessen wurde [SW97]. Gerade diese sequentiellen Simulationen haben die Notwendigkeit für eine Spezialhardware oder einen parallelen Ansatz aufgezeigt.

Mit dem *SpikeNET*-Simulator wird in [DGR99] eine Software vorgestellt, die neben der verfügbaren sequentiellen Implementierung eine parallele Umsetzung erlauben soll. Zweck des Simulators ist die Simulation großer Bildverarbeitungs-PCNN, wobei eine Simulationszeitreduktion durch ereignisbasierte Methoden zum Einsatz kommt. Zur parallelen Umsetzung wird auf den Austausch von Spikelisten verwiesen, Aussagen zur Lastverteilung oder Netzpartitionierung werden nicht gemacht. Die Autoren vermuten, bis zu 400.000 Neuronen mit 20 Mio. Verbindungen auf einem Apple PowerPC in 1 ms pro Zeitschritt simulieren zu können. Es werden keine Aussagen über tatsächlich simulierte Netze oder Modellneuronen gemacht, so daß zum jetzigen Zeitpunkt eine Beurteilung dieser Arbeiten schwerfällt. Die eigenen Untersuchungen sowie die Ergebnisse aus Marburg und Berlin deuten zumindest an, daß sehr optimistisch geschätzt wurde.

Eine parallele Simulatorimplementierung an der Universität Bonn in der Arbeitsgruppe von Prof. Anlauf zielt auf den Einsatz wesentlich aufwendigerer ereignisbasierter Algorithmen, als sie in der vorliegenden Arbeit verwendet wurden [GA98]. Dabei wurden Netze mit bis zu 120 Neuronen auf bis zu drei Maschinen simuliert, die Beschleunigung gegenüber einer Maschine reichte bis zum Faktor 2,3. Aufgrund der in Kap. 3.5 dargestellten Simulationen mit der Wiedervorlageliste erscheint der Einsatz solcher aufwendigen Verfahren für große Netze wenig sinnvoll. Im Fall sehr kleiner Netze und einer sehr feinen Zeitschritteinteilung, d.h. beim Vorliegen weniger Ereignisse pro Zeitschritt, kann jedoch ein Vorteil aufwendigerer Verfahren unterstellt werden.

Neben diesem Ansatz für kleine Netze wurden auch größere PCNN parallel simuliert. In [MSP97] wird die Simulation des PCNNs aus [SRE96] auf einem IBM SP2-Parallelrechner an der Universität Marburg beschrieben. Ein 16k Neuronen großes Netz dieses Typs erreicht auf einem SP2-Prozessor eine Simulationszeit von 141 ms pro Zeitschritt, auf acht Prozessoren werden 113 ms erreicht. Somit ist kaum eine Beschleunigung zu verzeichnen. Ein weiteres, konstruiertes Netz ohne praktische Bedeutung konnte hingegen auf acht Prozessoren etwa viermal schneller als auf einem Prozessor simuliert werden. Das Netz aus [SRE96] weist durchaus Ähnlichkeiten zum Weitzelnetz auf, weswegen die schlechte Beschleunigung verwundert. Auch das speziell konstruierte Netz weist schlechtere Beschleunigungswerte auf, als die im Rahmen dieser Arbeit durchgeführten Simulationen, obwohl ein Parallelrechner statt eines Workstationclusters verwendet wurde. Die Folgerung aus [MSP97], daß eine lagenbasierte Netzverteilung, d.h. ein horizontaler Schnitt, einer vertikalen säulenbasierten Methode vorzuziehen ist, kann nach den Untersuchungen mit dem Weitzelnetz nicht betätigt werden.

Als besonders gut zur PCNN-Simulation geeignet hat sich in [BN94][NB94] ein Thinking Machines CM2 Parallelrechner erwiesen. Diese Maschine verfügt über bis zu 65.536 einfache Prozessoren und ein sehr leistungsfähiges Verbindungsnetzwerk (Hypercube), mit dem jeder Prozessor jedem anderen über maximal zwölf Zwischenstationen eine Nachricht sen-

den kann. Die Prozessoren werden im SIMD-Modus betrieben, d.h. alle Prozessoren führen auf unterschiedlichen Daten die gleiche Operation aus. Zur Simulation wird eine empfängerorientierte Netztopologiespeicherung auf den Knoten des Zielneurons zusammen mit einer PreSpike-Übertragung an jeweils alle Prozessoren benutzt. Dieses Vorgehen kommt der SIMD-Arbeitsweise entgegen, da jeder Prozessor zu einem Spike eine Verbindung aufweist, deren Verbindungsgewicht (ggf. 0) er auf die Aktivität des von ihm simulierten Neurons akkumuliert. Auf der CM2 wurde kein reales Netz simuliert, sondern die Neuronen emittierten aufgrund einer Wahrscheinlichkeitsfunktion Spikes an die über eine pseudozufällige Verknüpfungsstruktur verbundenen Nachfolger. Somit stehen keine direkten Vergleichszahlen zur Verfügung. Auf einer 16k großen Prozessorgruppe wurden Netze mit 16k, 32k und 64k Neuronen simuliert, wobei jedes Neuron 1.024 Nachfolger hat. Die Simulationszeit stieg linear mit der Anzahl der Neuronen pro Prozessor, so daß vermutet wurde, daß dieses Verhalten bis zu 4 Mio. Neuronen auftritt. Die Simulationen auf der CM2 stellen sicherlich die feinstkörnigste Parallelisierung der PCNN-Simulation dar. Leider stehen keine Meßdaten für reale Netze zur Verfügung. Die Maschine wird zudem nicht mehr gebaut.

Der vorgenommene Vergleich mit in der Literatur dokumentierten PCNN-Simulationen zeigt, daß die im Rahmen dieser Arbeit untersuchten Netze die größten bisher simulierten Netze sind. In dieser Größenordnung sind zudem bisher keine vergleichbaren Geschwindigkeitszuwächse bei der Parallelisierung erreicht worden. Der implementierte Softwaresimulator erreicht insbesondere auch bessere Geschwindigkeitszuwächse als andere parallele Simulationen konventioneller neuronaler Netze. Die Skalierbarkeit ist zudem schon auf dem Workstationcluster besser als es bei anderen Simulationen selbst auf Parallelrechnern der Fall ist. Damit zeigen sich die ermittelten Simulationsverfahren den bisherigen überlegen.

Mit dem ParSPIKE-Konzept tritt die parallele PCNN-Simulation in Konkurrenz zu speziellen Neuroprozessoren wie dem SPIKE128k oder dem NESPINN/MASPINN. Auch in diesem Vergleich können ebenbürtige Simulationsleistungen prognostiziert werden. Damit ist die grundsätzliche Eignung der PCNN-Simulation zur Parallelisierung deutlich geworden. Zu klären ist demnach, wann sie anderen Ansätzen überlegen ist, d.h. unter welchen Randbedingungen welcher Ansatz zur Anwendung kommen sollte.

7.5 Effizienzbetrachtung

Nachdem nunmehr die Rahmendaten des ParSPIKE-Konzeptes und die Parameter der wichtigsten Vergleichssysteme dargestellt sind, sollen Aussagen über den grundsätzlichen Sinn einer ParSPIKE-Implementierung gemacht und das System bezüglich der Anwendung positioniert werden. In diese Betrachtungen wird auch der implementierte PVM-Softwaresimulator einbezogen.

Grundsätzlich lassen sich zwei Anforderungsfelder an einen PCNN-Simulator definieren. Zur *Entwicklung von PCNN* und für grundsätzliche Untersuchungen über ihre Arbeitsweise und ihren Aufbau ist ein System geeignet, das möglichst flexibel in bezug auf Änderungen der Modelle, Netze und Simulationsverfahren ist und eine einfache Überwachung der Simulationsparameter erlaubt. Die reine Simulationsgeschwindigkeit ist hier weniger relevant, die simulierbare Netzgröße muß jedoch ausreichen. Zur Untersuchung und *Evaluierung von PCNN-Applikationen in realen Szenarien* ist dagegen eine der Realzeitanforderung von 1 ms pro Zeitschritt [DDW98] nahekommende Simulationsleistung notwendig. Zudem ist ein kompakter Aufbau und ein geringer Energiebedarf z.B. in mobilen Anwendungen ein weiteres wichtiges Kriterium. Für beide Anwendungsfälle gilt, daß ein übersichtlicher und einfacher Aufbau die Implementierung eines Prototypsystems in einem universitären Umfeld erst ermöglicht. In diesem Spannungsfeld sind ParSPIKE und PVM-Simulator einzuordnen. Insbesondere die Einordnung im Vergleich mit NESPINN/MASPINN und SPIKE128k sowie eine ausführliche Beschreibung der Architektur dieser Systeme sind in [SSW00] zu finden.

Bezüglich der *Simulationsleistung* übertrifft der PVM-Simulator andere Softwaresimulatoren. Durch die Parallelisierung ist er in der Lage, auf einem Netzwerk von Standardrechnern eine Netzgröße zu simulieren, die auf einer einzelnen Maschine schon aus Speichergründen nicht unterzubringen ist. Zudem ist er auf einer großen Palette von Parallelrechnern lauffähig. Er kann jedoch in der Simulationsgeschwindigkeit nicht mit dem ParSPIKE konkurrieren. Der Grund dafür liegt in der schlechten Unterstützung der kleinteiligen Kommunikation durch lokale Rechnernetze oder Parallelrechner, da dort zumindest ein Teil des Aufwandes in Software abgewickelt werden muß. Das ParSPIKE-System kombiniert hingegen kommerzielle Prozessoren mit einer speziellen Kommunikationshardware und erreicht damit Geschwindigkeiten, die selbst von sehr viel aufwendigeren Parallelrechnern nicht übertroffen werden. Die Simulationsleistung eines einzelnen nrc-VME-Moduls mit 16 ADSP21160 übertrifft immerhin die Simulationsleistung einer Sun Ultra 60 um den Faktor 11. Allgemeiner gehaltene DSP-Hardware erreicht auf Basis eines VME-Moduls ebenfalls bei weitem nicht die ParSPIKE-Leistung, da die Integration geringer und die Kommunikationsleistung schlechter ist. Durch den Rückgriff auf kommerzielle Prozessoren profitiert das ParSPIKE-Konzept zudem von der Weiterentwicklung der Digitaltechnik, so daß der Leistungsunterschied zu konventionellen Rechnern auch in Zukunft gegeben ist. Insbesondere der Ausblick auf die weiteren SHARC-Modelle und die Verfügbarkeit von alternativen Rechenknoten mit großem on-Chip-Speicher lassen ein Erreichen der benötigten Geschwindigkeiten erwarten. Im Vergleich zu Spezialprozessoren wie dem MASPINN ergibt sich, daß diese ASICs auf Basis eines Rechenknotens mit vier Berechnungspipelines die Leistung von 64 ADSP21160-Prozessoren erreichen und somit wesentlich weniger Parallelisierungsaufwand benötigen. Allerdings zeigt der SPIKE128k, dessen Leistung durch 8 ADSP21160 übertroffen wird, daß dieser Vorteil nur von kurzer Dauer ist, da eine universitäre Prozessorentwicklung sich nicht automatisch mit dem kommerziellen Halbleitermarkt weiterentwickelt.

Neben der reinen Simulationsleistung ist auch der *Ressourcenbedarf* einer Implementierung von Belang. Zunächst scheinen hier die Einprozessor-Lösungen des SPIKE128k und des NESPINN/MASPINN die günstigste Variante zu bieten. Der SPIKE128k-Prozessor ist jedoch über viele Bauteile verteilt, so daß der Gesamtaufbau den Umfang eines kompletten VME-Systems erreicht. NESPINN integriert diese Leistung auf einer VME-Karte, MASPINN simuliert 1 Mio. Neuronen auf einer PCI-Platine. Dabei ist zu beachten, daß ein NESPINN/MASPINN-Prozessor nur die Neuronenberechnungen durchführt. Zur Speicherung der Neuronendaten und zur Verarbeitung der Netztopologie sind weitere Einheiten notwendig, deren Aufwand jedoch so gering eingeschätzt wird, daß ein NESPINN/MASPINN-Prozessor mit diesen Komponenten auf einem VME- bzw. PCI-Modul Platz findet. ParSPIKE benötigt für 1 Mio. Neuronen mindestens zwei VME-Module, für 128k Neuronen immerhin ein PCI-Modul. Auch bezüglich des Energiebedarfs dürfte ein MASPINN-Modul günstiger abschneiden als die ParSPIKE-Module mit 64 DSPs, wobei diese wiederum günstiger als das SPIKE128k-System einzuordnen sind. Der PVM-Simulator erreicht auf vier Sun Ultra 60 die Simulationsleistung des SPIKE128k und auf 32 dieser Maschinen möglicherweise die MASPINN- bzw. ParSPIKE-Leistung, so daß die Softwaresimulation den sicherlich höchsten Ressourcenbedarf aufweist. Ein voll ausgebautes ParSPIKE-System mit 8 VME-Modulen bzw. ein PC mit 4 MASPINN-PCI-Modulen übertrifft die Simulationsleistung eines Parallelrechners mit 128 modernen Prozessoren. Zum Betrachtung des Ressourcenbedarfs einer Implementierung gehört sicherlich auch der finanzielle Aufwand. Dabei ist der Zeitaufwand vom Materialaufwand zu trennen, da in universitären Projekten der Zeiteinsatz auch zu Lern- und Erkenntniszwecken erbracht wird. Bei der Realisierung eines Prototypsystems einer Spezialhardware bildet jedoch Zeiteinsatz den wesentlichen Beitrag, so daß eine Kostenbetrachtung in diesem Zusammenhang nicht sinnvoll erscheint.

Bezüglich der *Flexibilität* ist hingegen die Softwaresimulation den Spezialhardwaresystemen überlegen, da sie fast beliebige Anpassungen und Protokollausgaben aller Art mit geringem Aufwand zuläßt. ParSPIKE ist in diesem Bereich MASPINN überlegen, da wesentliche Teile der Simulation in Software abgebildet sind. Die Programmierung ist jedoch aufgrund der engen Rahmenbedingungen des SHARC-DSPs und der teilweisen Assemblercodierung recht aufwendig. Zur Flexibilität des ParSPIKE-Konzeptes trägt bei, daß es sich auf andere kommerzielle Prozessoren übertragen läßt. Die Spezialhardwarekomponenten bestehen aus FPGAs und lassen sich daher umprogrammieren. MASPINN erlaubt hingegen nur eine gewisse Programmierbarkeit des Modellneurons.

Der *Entwicklungsaufwand* ist bei den Softwaresimulatoren wesentlich geringer als bei jedweder Spezialhardware. Auch die *Wartbarkeit* eines Softwarepakets stellt sich einfacher dar. Somit ist eine Softwarelösung die am einfachsten zu implementierende Simulatorvariante. Gerade deshalb wurde mit dem ParSPIKE versucht, soviel Entwicklungsaufwand des Simulators durch kommerzielle Prozessoren und Software abzudecken wie möglich. Zudem wur-

de für die Spezialhardwarekomponenten auf programmierbare Logik zurückgegriffen, deren Test und Inbetriebnahme einfacher ist als dieses bei ASICs der Fall ist.

Die *Handhabbarkeit* der Simulatoren spricht für den PVM-Simulator, da in fast jedem Entwicklungslabor eine geeignete Simulationsplattform vorhanden ist und der Einarbeitungsaufwand durch die Verwendung einer gewohnten Umgebung gering ist. Die MASPINN- und ParSPIKE-Module basieren auf verbreiteten Systemschnittstellen wie dem PCI- oder VME-Bus und lassen sich daher in ggf. vorhandene Systemumgebungen integrieren. Bezüglich der Fehleranalyse ist der ParSPIKE überlegen, da für die DSPs entsprechende Werkzeuge existieren. Allerdings verlangt die parallele Implementierung sowohl des ParSPIKE als auch des PVM-Simulators einen entsprechenden Compiler, der wiederum sehr aufwendig ist und maßgeblichen Einfluß auf die erreichbare Leistung hat. Dagegen bedarf es beim MASPINN bis zu Netzgrößen von 1 Mio. Neuronen keinerlei Partitionierungsüberlegungen.

Die bisherigen Betrachtungen haben gezeigt, daß sich die Einordnung des ParSPIKE an zwei Vergleichssystemen orientiert, dem PVM-Simulator auf den diversen Plattformen und der Spezialhardware MASPINN. Alternative Softwaresimulatoren sind weniger leistungsfähig, NESPINN und SPIKE128k sind als Vorläufer des MASPINN einzuordnen.

Der PVM-Simulator bietet eine unerreichte Flexibilität und die Möglichkeit des Einsatzes auf weitgehend verfügbaren Plattformen. Er kann jedoch in bezug auf den Ressourcenbedarf und die Simulationsleistung nicht mit einer Spezialhardware mithalten. Das PVM-System bietet sich damit für die zeitunkritische PCNN-Entwicklung an, da es in vorhandenen Umgebungen die dazu benötigte Leistung bereitstellt und einfach zu handhaben ist. Zu einem vollständigen PCNN-Entwicklungssystem ist allerdings auch der MNET-Compiler notwendig, der zu diesem Zweck sicherlich optimiert werden muß, da es wenig sinnvoll ist, den Simulationsteil des Entwicklungszyklus auf einige Minuten zu verkürzen, wenn der ebenfalls notwendige Compilierungsvorgang etliche Stunden dauert. Weiterhin sind entsprechende Visualisierungs- und Debbuging-Werkzeuge unerläßlich. Die Integration unter einer ergonomischen Benutzungsoberfläche erleichtert die Entwicklung zusätzlich.

ASIC-Entwicklungen wie der MASPINN bieten die günstigste Lösung, wenn eine konkrete PCNN-Applikation in ein reales Szenario eingebettet werden soll. Der Grad der Integration übertrifft das ParSPIKE-Konzept, der Energiebedarf ist geringer und die Problematik der Parallelisierung entfällt weitgehend. Der Nachteil liegt in dem hohen Entwicklungsaufwand und in der Tatsache, daß ein universitärer Spezialprozessor kaum von der schaltungstechnischen Weiterentwicklung profitiert und schnell veraltet. Die Entwicklung eines Prozessor-ASICs schließt allerdings die Parallelisierung nicht aus. Eine Integration eines speziellen Neuro-Prozessors in das ParSPIKE-System ist als Ersatz für den SHARC-DSP möglich und würde zu einer maximalen Simulationsleistung führen. Die hohe Leistung würde allerdings durch die Verbindung von Entwicklungsaufwand und Parallelisierungsaufwand erkauft.

Das ParSPIKE-System stellt einen Kompromiß zwischen der programmtechnischen Simulation und der schaltungstechnischen Realisierung dar. Wie gezeigt wurde, ist es im Zuge der Entwicklung im Prozessorbereich sehr wohl in der Lage, die Anforderungen zu befriedigen und bei etwas höherem Platz- und Energiebedarf die Leistung eines MASPINN-Systems zu bieten. Diese Leistung wird mit einem geringeren Implementierungsaufwand erreicht. Die zudem höhere Flexibilität bezüglich Änderungen macht den ParSPIKE insbesondere als Prototypsystem in einem universitären Umfeld attraktiv. Einer solchen Prototypimplementierung kommt auch die Verwendung der programmierbaren FPGAs bezüglich Test und Inbetriebnahme entgegen, wobei die Beschreibung dieser Bausteinentwürfe in einer Hochsprache (VHDL) prinzipiell auch die Umsetzung auf schnellere ASICs ermöglicht. Bei gegenüber dem MASPINN reduzierten Leistungsanforderungen ist eine Integration des ParSPIKE in ein reales Applikationsszenario denkbar. Insbesondere die Flexibilität in bezug auf den Rechenknoten läßt zudem auch in Zukunft eine ausreichende Simulationsleistung erwarten. Dazu muß jedoch vor einer Implementierung eine erneute Prüfung des sehr dynamischen Prozessormarktes erfolgen, um mögliche Alternativen in Betracht zu ziehen und das Konzept ggf. anzupassen.

Der bisher vorgesehene SHARC-DSP kann in den erhältlichen Ausführungen ADSP21060 und ADSP21160 bezüglich der Rechengeschwindigkeit die Anforderungen nicht vollständig befriedigen. Dabei ist zu beachten, daß diese Anforderungen durch die Größe des on-Chip-Speichers vorgegeben sind, d.h. bei wachsendem on-Chip-Speicher ist auch zusätzliche Rechengeschwindigkeit wünschenswert. Gerade im Rahmen der SHARC-Reihe wächst jedoch die Rechengeschwindigkeit überproportional. Während noch beim ADSP21060 die durch den Speicher mögliche Simulationsleistung etwa zwanzigmal höher als die tatsächlich erreichte Leistung lag, wird dieses Verhältnis beim ADSP21160 auf den Faktor acht und beim TigerSHARC etwa auf den Faktor zwei reduziert.

Ein weiterer Vorteil der Implementierung des ParSPIKE auf Basis von DSPs liegt zudem darin, daß solche Prozessoren auch für andere Aufgaben benutzt werden können. Dazu kann z.B. wie in Kap. 7.3 vorgeschlagen eine Bildvorverarbeitung in Form von Gangliendecodern implementiert werden, die aus über den VME-Bus empfangenen Bildern Spikes erzeugt. Eine weitere Möglichkeit zur Nutzung der DSPs besteht in der Parallelisierung von Teilen des MNET-Compilers, z.B. zur Berechnung der Verknüpfungsmasken aus den mathematischen Beschreibungen.

Insgesamt empfiehlt sich der Aufbau eines ParSPIKE-Systems dann, wenn die Simulationsleistung des PVM-Simulators nicht mehr ausreicht und der Aufwand und die fehlende Flexibilität einer Spezialprozessorentwicklung gescheut werden. Die Parallelisierung der Simulation an sich bietet jedoch für alle diese Möglichkeiten sinnvolle Perspektiven.

Kapitel 8 - Zusammenfassung und Ausblick

Große pulscodierte neuronale Netze (PCNN) für die Bildverarbeitung stellen ein Simulationsproblem dar, das nur mit erheblicher Rechenleistung in akzeptabler Zeit lösbar ist. Ziel der vorliegenden Arbeit war die Untersuchung der Frage, ob ein paralleler Simulationsansatz die benötigte Leistung zur Verfügung stellen kann. Insbesondere vor dem Hintergrund, daß Simulationen konventioneller neuronaler Netze aufgrund des hohen Kommunikationsanteils und des vergleichsweise geringen Rechenaufwands schlecht zu parallelisieren sind [Str94] [Res99], galt es festzustellen, ob der durch das komplexere Modellneuron verursachte höhere Rechenaufwand und die durch die pulscodierte Übertragung geringere Kommunikationslast zu einer Beschleunigung der Simulation durch die Parallelisierung führen. Auslöser dieser Untersuchungen war die Fragestellung, ob mit einem parallelen System auf Basis kommerzieller Prozessoren die Simulationsleistung eines wesentlich aufwendiger zu implementierenden und inflexibleren Spezialprozessorsystems erreicht werden kann.

Da pulscodierte neuronale Netze deutlich andere Anforderungen an die Simulation stellen, als konventionelle neuronale Netze, wurde im zweiten Kapitel der Arbeit zunächst die Funktionsweise von PCNN erläutert und den konventionellen Neuronenmodellen gegenübergestellt. Als Grundlage der weiteren Simulationen wurden Beispielnetze aus dem Bereich der Bildverarbeitung vorgestellt und an diesen Beispielen die für die Simulation relevanten Charakteristika der PCNN herausgearbeitet. Die Netze zeichnen sich durch einen systematischen Aufbau, eine spärliche Verknüpfung, eine geringe Aktivität und eine niedrige Spikerate aus.

Insbesondere die geringe Aktivität und die niedrigen Spikeraten legen für die digitale Simulation die Verwendung eines ereignisbasierten Ansatzes nahe. Die grundlegenden Techniken und ihre Anwendung auf die PCNN wurden im dritten Kapitel erläutert. Es zeigt sich, daß die ereignisbasierte Simulation [Fer95] zu einer drastischen Reduktion der Rechenschritte führt. Aufgrund der sehr hohen Anzahl von Ereignissen in der PCNN-Simulation ermöglichen jedoch nur sehr einfache ereignisbasierte Verfahren eine Simulationsbeschleunigung, während der Vorteil weitergehender Verfahren durch den zusätzlichen algorithmischen Aufwand verschwindet. Mit einem sequentiellen Softwaresimulator wurden die Beispielnetze untersucht. Kommerzielle Standard-Rechner bieten dazu, wie in [JRK95][RJK95] postuliert,

keine ausreichende Simulationsleistung. Gefordert ist, Netze mit 1 Mio. Neuronen in 1 ms pro Zeitschritt [DDW98] zu simulieren, Standard-Rechner verfehlen dieses Ziel um mindestens zwei Größenordnungen. Im Rahmen der Untersuchung der sequentiellen ereignisbasierten Simulation erfolgte auch die Vorstellung des SPIKE128k-Systems [Fra97], welches einige ereignisbasierte Algorithmen in einer Spezialprozessoreinheit integriert. Der immense Entwicklungsaufwand für diesen Simulator führt gerade zu dem Wunsch, eine vergleichbare Leistung mit einem parallelen System auf Basis kommerzieller Prozessoren zu erzielen. Da gewöhnlich nicht für jedes Neuron ein eigener Rechenknoten bereitsteht, erfolgt die Simulation auf den parallelen Rechenknoten sequentiell mit Hilfe der ereignisbasierten Verfahren. Dabei finden die im SPIKE128k verwendeten Ansätze der Spikeliste und der Abklingliste Anwendung, während von den neu erarbeiteten Ansätzen der EIB-Cache als vielversprechend beibehalten und das aufwendige Wiedervorlageverfahren verworfen wurde.

Die Parallelisierung des sequentiellen Simulationsverfahrens bildet den Schwerpunkt des vierten Kapitels. Neben den grundsätzlichen Verfahren zur Synchronisation und Ablaufsteuerung wurden insbesondere Fragestellungen der Kommunikation, der Darstellung der Verknüpfungsstruktur des Netzes und der Verteilung der Neuronen auf die Rechenknoten behandelt und in den Kontext anderer Ansätze gestellt. Zur Kommunikation findet eine Übertragung der präsynaptischen Spikes (PreSpikes) zwischen den Rechenknoten Anwendung, da die PreSpikes die geringstmögliche zu übertragende Datenmenge bilden. Zudem ist dieses Vorgehen biologisch plausibel, denn in Gehirnen wird ein Spike über ein einziges Axon ins Zielgebiet transferiert und erst dort über die Synapsen verteilt. Für die Darstellung der Netztopologie wurde ein Ansatz mit einer zentralen Speicherung der Netztopologie und Cache-Speichern auf den Rechenknoten erläutert, der insbesondere für speichergekoppelte parallele Systeme (shared memory) geeignet erscheint. Solche Systeme existieren als Parallelrechner in Form der symmetrischen Multiprozessoren (SMP). Ein weiterer Ansatz verteilt die Netztopologie auf die Rechenknoten, was zu einem nachrichtengekoppelten parallelen Rechenparadigma führt (message passing) und insbesondere der Architektur massiv paralleler Systeme entgegenkommt. Für Systeme mit begrenztem Rechenknotenspeicher erfolgte die Präsentation einer Variante auf Basis einer kompakten Topologiedarstellung mit regulären Verknüpfungsmasken. Bei der Lastverteilung besteht aufgrund der unbekannten Aktivitätsverteilung im Netz, die durch einen unbekannten Eingangsreiz verursacht wird, eine Konkurrenz zwischen einem niedrigen Verbindungsschnitt und einer guten Lastverteilung, die das Aufbrechen von Aktivitätsschwerpunkten erfordert. Vor diesem Hintergrund wurden verschiedene Klassen von Partitionierungsverfahren vorgestellt.

Das fünfte Kapitel befaßt sich mit der Implementierung der parallelen Simulationsmethoden. Dazu wurde das eigentliche Simulationsverfahren auf Basis des PVM (Parallele Virtuelle Maschine) Systems [GBD94] auf einer Gruppe von Workstations umgesetzt, die mit einem lokalen Netz verbunden sind (Workstationcluster). Die Partitionierungsverfahren sind in

Form eines Compilers für die Neuronale-Netze-Sprache MNET exemplarisch implementiert. Die durchgeführten Simulationen mit den im zweiten Kapitel beschriebenen Beispielen Weitzelnetz und Brausenetz zeigen das grundsätzliche Potential der parallelen PCNN-Simulation. Das Weitzelnetz wird auf acht Maschinen fast sechsmal schneller simuliert als auf einer Maschine, das Brausenetz erreicht selbst beim Übergang von acht auf zwölf Maschinen einen Geschwindigkeitszuwachs um den Faktor 1,45 und liegt damit nahe am theoretischen Maximum. Diese guten Ergebnisse entstanden nicht auf einem Parallelrechner mit entsprechenden Kommunikationsressourcen, sondern auf einem wesentlich schlechter geeigneten Workstationcluster, der über ein 10 MBit/s schnelles Ethernet verknüpft ist. Schon auf dieser Plattform übertrifft das Ergebnis der Parallelisierung alle vergleichbaren Simulationen. Zur Erreichung dieser Geschwindigkeitszuwächse zeigte sich eine Netzaufteilung als sinnvoll, die aus den übereinandergelegten Neuronenschichten zusammenhängende Säulen ausschneidet. Dabei werden wenige der sehr lokalen Verknüpfungsschemata zerschnitten und gleichzeitig Aktivitätsschwerpunkte in den Neuronenschichten aufgebrochen.

Trotz der guten Ergebnisse auf dem Workstationcluster ermöglicht der implementierte Softwaresimulator nicht die Erreichung der gewünschten Geschwindigkeiten. Insbesondere bei hohen Prozessorzahlen ist das lokale Netzwerk nicht in der Lage, die sehr kleinteilige Kommunikation abzuwickeln. Auch andere Kommunikationsnetzwerke, deren Protokolle zumindest zu Teilen auf Software basieren, sind hierfür ungeeignet. Daher erfolgte im sechsten Kapitel die Vorstellung eines Systems auf Basis kommerzieller Prozessoren, welches über ein spezielles Kommunikationssystem den Spikeaustausch in Hardware abwickelt. Für dieses ParSPIKE-System wurden Hardware-Komponenten in Form programmierbarer Bausteine entwickelt (FPGAs). Als Rechenknoten findet ein digitaler Signalprozessor (DSP) der Firma Analog Devices aus der SHARC-Reihe Verwendung, der über einen großen Speicher und eine Multiprozessorunterstützung auf dem Chip verfügt. Zur Realisierung des ParSPIKE-Systems wurden Vorschläge auf Basis der VME- und PCI-Busschnittstelle gemacht.

Im siebten Kapitel erfolgte eine Leistungsabschätzung für das gesamte ParSPIKE-System in verschiedenen Szenarien. Dazu sind die ausgewählten DSPs mit den Verfahren programmiert und optimiert worden. Zusätzlich zu den Evaluierungswerkzeugen für die Signalprozessoren wurde ein Testsystem auf Basis von zwei DSPs und einem FPGA implementiert und vermessen. Die Abschätzungen zeigen, daß auf Basis des Analog Devices ADSP21160 für ein Netz mit 1 Millionen Neuronen eine Simulationsgeschwindigkeit von bis zu 8 ms pro Zeitschritt erreicht werden kann. Zu diesem Zweck ist ein System mit 64 Prozessoren erforderlich, das je nach Auslegung auf 2-3 VME-Bus-Modulen Platz findet. In der Perspektive lassen die Nachfolger des ADSP21160 aus der TigerSHARC-Reihe Zeiten bis zu 3,2 ms zu. Neben der Leistungsbewertung des ParSPIKE erfolgte zudem der Vergleich mit Alternativsystemen. Insbesondere die Auswahl an Rechenknoten auf dem kommerziellen Prozessormarkt läßt erwarten, daß der ParSPIKE von zukünftigen Entwicklungen profitiert. Gerade

darin lag eines der Hauptziele einer Parallel-Hardware auf Basis von kommerziellen Komponenten. Das ParSPIKE-System ermöglicht zudem eine höhere Leistung, als sie mit anderen Parallelrechnern oder DSP-Systemen sowie mit konventionellen Neurocomputern zu erreichen ist. Einzig die PCNN-Spezialrechner auf Basis der MASPINN-Prozessoren [SMJ98], welche die SPIKE128k-Architektur aufgegriffen und weiterenwickelt haben, sind in der Lage, die Leistungen des ParSPIKE zu erreichen und in bezug auf die Platz- und Energieeffizienz z.T. zu übertreffen. Auf Basis eines Systems mit einem MASPINN-Prozessor ist für 1 Mio. Neuronen eine Simulationszeit von 6,4 ms pro Zeitschritt zu erreichen.

Im Kontext der verschiedenen Realisierungen läßt sich der PVM-Simulator aufgrund seiner überlegenen Flexibilität und der Eignung für eine Vielzahl von Simulationsplattformen als ideales Instrument für die Entwicklung und grundlegende Untersuchung von PCNN positionieren. Ein Spezialchip wie der MASPINN ist dagegen für die zeitkritische PCNN-Applikation in realen Szenarien geeignet, wobei der Entwicklungsaufwand immens und die Überlegenheit über kommerzielle Prozessoren kurzlebig ist. Dagegen bietet das ParSPIKE-Konzept einen Kompromiß, der die benötigte Simulationsleistung bei höherem Ressourcenbedarf und niedrigerem Entwicklungsaufwand bereitstellt. Die Flexibilität liegt durch die Abbildung von Teilen des Simulators in Software zwischen einer reinen Software-Lösung wie dem PVM-Simulator und einer reinen Hardware-Implementierung wie dem MASPINN.

Damit ist der parallele Ansatz für die PCNN-Simulation sowohl als Software-Lösung als auch als Spezialhardware geeignet. Die vorliegende Arbeit hat einen Überblick zu den wesentlichen Fragen gegeben sowie exemplarisch Lösungen und weitere Tendenzen aufgezeigt. Die nunmehr gegebenen ausreichenden Simulationsmöglichkeiten ermöglichen die Entwicklung und Untersuchung weiterer Beispielnetze und damit die Verifikation der gemachten Aussagen. Weiterhin eröffnen die Suche nach alternativen Methoden der Netzpartitionierung und die Optimierung des Neuronale-Netze-Compilers Möglichkeiten für zukünftige Arbeiten. Dadurch wird eine leistungsfähige Entwicklungsumgebung für PCNN geschaffen, für die zudem Werkzeuge zur Visualisierung und Fehlersuche sowie eine ergonomische Benutzungsoberfläche zu implementieren sind. Arbeiten zur Weiterentwicklung des ParSPIKE-Konzepts finden auf dem zukünftigen Prozessormarkt genügende Ausgangspunkte, zudem können bei steigender Simulationsleistung aufwendigere Kommunikationsstrukturen eingesetzt werden, die in der Informatik in großer Zahl bekannt sind.

Insgesamt zeigen sich damit drei Stoßrichtungen, um die nachgewiesene Eignung der PCNN für eine parallele Simulation in sinnvolle Applikationen umzusetzen. Zuerst ist eine leistungsfähige Entwicklungsumgebung erforderlich, zu der mit dem PVM-Simulator und dem MNET-Compiler die zentralen Grundsteine gelegt sind. Darauf aufbauend kann die Entwicklung wirklich überlegener PCNN-Anwendungen erfolgen. Zur Evaluierung und zum Einsatz dieser Anwendungen ist dann die Bereitstellung einer Spezialhardware notwendig, für die auf das beschriebene ParSPIKE-Konzept zurückgegriffen werden kann.

Literaturverzeichnis

ABC92 Arendt, F., Brinkkötter, W., Cordes, N., Dittmer, G., Juhasz, R., Reetz, A., Schäffer, M., Strasdat, H., Strothmann, W.B.: *Projektgruppe Simulation Neuronaler Netze/Zwischenbericht.* Interner Bericht, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (1992)

ABC93 Arendt, F., Brinkkötter, W., Cordes, N., Dittmer, G., Juhasz, R., Reetz, A., Schäffer, M., Strasdat, H., Strothmann, W.B.: *Projektgruppe Simulation Neuronaler Netze/Abschlußbericht.* Interner Bericht, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (1993)

AD96 Analog Devices: *ADSP-2106x SHARC User‘s Manual.* Second Edition 7/96, Analog Devices (1996)

AD00 Analog Devices: *ADI - DSP Technology Center.* www.analog.com (2000)

Bar72 Barlow, H.B.: *Single Units and Sensation: A Neuron Doctrine for Perceptual Psychology.* Perception, Vol. 1, pp. 371-394 (1972)

BB94 Bower, J.M., Beeman, D.: *The Book of GENESIS.* Springer Verlag (1994)

BN94 Brettle, D., Niebur, E.: *Detailed Parallel Simulation of a Biological Neuronal Network.* Computational Science and Engineering I, pp. 31-43 (1994)

Cib99 Cibula, A.: *Parallele Simulation biologienaher pulscodierter neuronaler Netze mittels PVM.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (1999)

CM98 Choe, Y., Miikkulainen, R.: *Self-Organisation and Segmentation in a Laterally Connected Orientation Map of Spiking Neurons.* Neurocomputing, Vol. 21, pp. 139-158 (1998)

Dau80 Daugman, J.G.: *Two-Dimensional Spectral Analysis of Cortical Receptive Field Profiles.* Vision Research, Vol. 20, pp. 847-856 (1980)

DDW98 Deiss, S.R., Douglas, R.J., Whatley, A.M.: *A Pulse-Coded Communications Infrastructure for Neuromorphic Systems.* In: Maass, W., Bishop, C.M.: Pulsed Neural Networks. A Bradford Book, MIT Press, pp. 155-178 (1998)

Des37 Descartes, R.: *Discours de la Méthode.* (1637)

DGR99 Delorme, A., Gautrais, J., van Rullen, R., Thorpe, S.: *SpikeNET: A simulator for modeling large networks of integrate and fire neurons.* Neurocomputing 26-27, Elsevier Science B.V., pp. 989-996 (1999)

Dit94 Dittmer, G.: *Lernverfahren für neuronale Netze und ihre Implementierung auf Message-Passing Multicomputern.* Diplomarbeit, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (1994)

DPG94 Deppisch, J., Pawelzik, K., Geisel, T.: *Uncovering the synchronization dynamics from correlated neural activity quantifies assembly formation.* Biological Cybernetics 71, pp. 387-399 (1994).

EBJ88 Eckhorn, R., Bauer, R., Jordan, W., Brosch, M., Kruse, W., Munk, M., Reitböck, H.J.: *Coherent Oscillations: A Mechanism of Feature Linking in the Visual Cortex?* Biological Cybernetics, 60, pp. 121-130 (1988)

EB00 Eyre, J., Bier, J.: *The Evolution of DSP Processors - From Early Architectures to the Latest Developments.* IEEE Signal Processing Magazine, March 2000, pp. 43-51 (2000)

Eck94 Eckhorn, R.: *Oscillatory and Non-Oscillatory Synchronizations in the Visual Cortex and their Possible Roles in Associations of Visual Features.* Progress in Brain Research, 102, pp. 405-426 (1994)

EDA92 Eckhorn, R., Dicke, P., Arndt, M., Reitböck, H. J.: *Feature linking of visual features by stimulus-related synchronizations of model neurons.* In: E. Basar, T. H. Bullock (ed.), Induced Rhythms in the Brain, Brain Dynamics Series, Brinkhäuser, Boston, Basel, Berlin, pp. 140-154 (1992).

Ehl99 Ehlert, M.: *Analoge Schaltungstechnik für die Emulation biologieorientierter Neuronenmodelle*. Dissertation, Technische Universität Berlin, Shaker-Verlag (Berichte aus der Elektrotechnik), Aachen (1999)

EKG90 Engel, A.K., König, P., Gray, C.M., Singer, W.: *Stimulus-Dependent Neuronal Oscillations in Cat Visual Cortex: Inter-Columnar Interaction as Determined by Cross Correlation Analysis.* European Journal of Neuroscience, 2, pp. 588-606 (1990)

ERA89 Eckhorn, R., Reitböck, H. J., Arndt, M., Dicke, P.: *Feature Linking via Stimulus - Evoked Oscillations: Experimental Results for Cat Visual Cortex and Functional Implications from a Network Model.* Proc. IJCNN89, Vol. I, pp. 723-730 (1989)

ERA90 Eckhorn, R., Reitböck, H. J., Arndt, M., Dicke, D.: *Feature Linking via Synchronization among Distributed Assemblies: Simulations of Results from Cat Visual Cortex*. Neural Computations 2, pp. 293-307 (1990),

FBH95 Frank, G., Bilau, N., Hartmann, G.: *Hardware Accelerator zur Simulation pulscodierter Neuronaler Netze.* DAGM 1995, Bielefeld. In: Sagerer, G; Posch, S.; Kummert, F. (Hg.): Mustererkennung 1995. Informatik aktuell. Berlin u.a., Springer Verlag, pp. 194-201 (1995)

Fer95 Ferscha, A.: *Parallel and Distributed Simulation of Discrete Event Systems.* In: Parallel and Distributed Computing Handbook, McGraw-Hill (1995)

FH95 Frank, G., Hartmann, G.: *An Artificial Neural Network Accelerator for Pulse-Coded Model Neurons.* ICNN95, Perth, Australia. In: Proceedings on International Conference on Neural Networks, Perth, Vol. 4, pp. 2014-2018 (1995)

FHJ99 Frank, G., Hartmann, G., Jahnke, A., Schäfer, M.: *An Accelerator for Neural Networks with Pulse-Coded Model Neurons.* IEEE Transactions on Neural Networks, Special Issue on Pulse Coupled Neural Networks, Vol. 10, pp. 527-539. (1999)

Fly66 Flynn, M.J.: *Very High-Speed Computing Systems*. Proceedings of the IEEE, Vol. 54, No. 12, pp. 1901-1909, (1966)

Fra97 Frank, G.: *Ein digitales Hardwaresystem zur echtzeitfähigen Simulation biologienaher neuronaler Netze.* Dissertation, Universität-GH Paderborn, HNI-Verlagsreihe, Bd. 26, Prof. Dr. rer. nat. Hartmann (Hrsg.), Paderborn (1997)

FS70 French, A. S., Stein, R. B.: *A Flexible Analog Using Integrated Circuits*. IEEE Transactions on Bio-Medical Engineering, Vol. BME-17, No. 3, pp. 248-253 (1970)

FvD87 Freeman, W.J., van Dijk, B.W.: *Spatial Patterns of Visual Cortical Fast EEG During Conditioned Reflex in a Rhesus Monkey.* Brain Research, 422, pp. 267-276 (1987)

GA98 Graßmann, C., Anlauf, K.: *Distributed, Event Driven Simulation of Spiking Neural Networks.* Proc. of the International ICSC/IFAC Symposium on Neural Computation (NC‘98), ICSC Academic Press, pp. 100-105 (1998)

GBD94 Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.: *PVM: A Users‘ Guide and Tutorial for Networked Parallel Computing.* MIT Press (1994)

GHL92 Gray, R.W., Heuring, V.P., Levi, S.P., Sloane, A.M., Waite, W.M.: *Eli: A Complete, Flexible Compiler Construction System.* Communications of the ACM, 35(2), pp. 121.131 (1992)

GRH93 Gerstner, W., Ritz, R., v. Hemmen, J. L.: *A biologically motivated and analytically soluble model of collective oscillations in the cortex. I. Theory of weak lokking.* Biological Cybernetics 68, pp. 363-374 (1993).

GS89 Gray, C. M., Singer, W.: *Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex.* Proc. National. Academic. Sociaty USA, 86, pp, 1698-1702 (1989).

Har91 Hartmann, G.: *Hierarchical Neural Representation by Synchronized Activtity: a Concept for Visual Pattern Recognition.* Proc. WCDNN, Vol. 1, pp. 356-370 (1991)

HD90 Hartmann, G., Drüe, S.: *Self Organization of a Network Linking Features by Synchronization*. Parallel Processing in Neural Systems and Computers, Eckmiller, Hartmann, Hauske (Eds.), pp. 361-364 (1990).

HD94a Hartmann, G.; Drüe, S.: *Why Synchronization? An Attempt to Show Quantitative Advantages.* Proc. of World Congress on Neural Networks (WCNN '94). San Diego, Bd. 1, S. 581-586 (1994)

HD94b Hartmann, G.; Drüe, S.: *Synchronization-Based Complex Model Neurons.* Proc. of ICANN 94, Sorrento, Bd. 1, 138-141 (1994)

Heb49 Hebb, D.O.: *The Organization of Behaviour.* Wiley, New York (1949)

Hee95 Heemskerk, J. N. H.: *Neurocomputer for Brain-Style Processing. Design, Implementation and Application.* Ph. D. Thesis, Leiden University, Rijksuniverditeit Leiden (1995)

HFS97 Hartmann, G., Frank, G., Schäfer, M., Wolff, C.: *SPIKE128K - An Accelerator for Dynamic Simulation of Large Pulse-Coded Networks.* In: Proceedings of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary & Fuzzy Systems, Dresden 1997, pp.130-139

HH52 Hodgkin, A.L., Huxley, A.F.: *A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve.* Journ. Physiol., Vol. 117, pp. 500-544 (1952)

HMB92 Hamilton, A., Murray, A.F., Baxter, D.J., Churcher, S., Reekei, H.M., Tarassenko, L.: *Integrated Pulse Stream Neural Networks: Results, Issues, and Pointers.* IEEE Transactions on Neural Networks, 3, 385-393 (1992)

HMM98 Hansel, D., Mato, G., Meunier, C.: *On Numerical Simulations of Integrate-and-Fire Neural Networks.* Neural Computation, Vol. 10, pp. 467-483 (1998)

Hub90 Hubel, D. H.: *Auge und Gehirn: Neurobiologie des Sehens.* Bd. 20, Spektrum der Wissenschaft Verlagsgesellschaft, Speyer (1990)

HW62 Hubel, D.H., Wiesel, T.N.: *Receptive Fields, Binocular Interaction and Functional Architecture in the Cat‘s Visual Cortex.* Journ. Physiology, Vol. 160, pp. 106-154 (1962)

Ibe99 Ibers, C.: *Ein Kontroll-FPGA für die Simulation von pulscodierten neuronalen Netzen auf einem parallelen DSP-System.* Studienarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (1999)

Ibe00 Ibers, C.: *Ein parallelisierender Compiler für die Neuronale-Netze-Sprache MNET.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

JE97 Juergens, E., Eckhorn, R.: *Parallel Processing by a Homogeneous Group of Coupled Model Neurons Can Enhance, Reduce and Generate Signal Correlations.* Biological Cybernetics, 76, pp. 217-227 (1997)

Joc00 Jochheim, G.: *Ein Kommunikations-Switch mit DSP-Interface in FPGA-Technik für einen Parallelrechner.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

Joh94 Johnson, J.L.: *Pulse-Coupled Neural Nets: Translation, Rotation, Scale, Distortion, and Intensity Signal Invariance for Images.* Applied Optics, Vol. 33, pp. 6239-6253 (1994)

JP99 Johnson, J.L., Padgett, M.L.: *PCNN Models and Applications.* IEEE Transactions on Neural Networks, Special Issue on Pulse Coupled Neural Networks, Vol. 10, pp. 480-498 (1999)

JRK95 Jahnke, A., Roth, U., Klar, H.: *Towards Efficient Hardware for Spike-Processing Neural Networks.* World Congress on Neural Networks WCNN‘95, Washington, USA, II, pp.460-463 (1995)

JRK96 Jahnke, A., Roth, U., Klar, H.: *A SIMD/Dataflow Architecture for a Neurocomputer for Spike-Processing Neural Networks (NESPINN).* MicroNeuro 96, Lausanne, Switzerland, pp. 232-237 (1996).

JRS98 Jahnke, A., Roth, U., Schönauer, T.: *Digital Simulation of Spiking Neural Networks.* In: Pulsed Neural Networks, W. Maass and C. M. Bishop (Eds.), MIT Press, pp. 237-258, (1998)

JSR97 Jahnke, A., Schönauer, T., Roth, U., Mohraz, K., Klar, H.: *Simulation of Spiking Neural Networks on Different Hardware Platforms.* In: Artificial Neural Networks-ICANN97. Springer Verlag, Berlin, pp.1187-1192 (1997)

Kas90 Kastens, U.: *Übersetzerbau, Handbuch der Informatik, Bd. 3.3.* Oldenbourg (1990)

KL99 Kinser, J.M., Lindblad, T.: *Implementation of Pulse-Coupled Neural Networks in a CNAPS Environment.* IEEE Transactions on Neural Networks, Special Issue on Pulse Coupled Neural Networks, Vol. 10, pp. 584-590. (1999)

Köh90 Köhle, M.: *Neurale Netze.* Springer Verlag, Wien (1990)

Kre93 Kreimeier, B.: *Distributed Implementation of Neural Networks: Lower Bounds of Communication Load.* Proceedings of the WCNN‘93, vol. IV, pp. 844-847 (1993)

KS91 König, A., Schillen, T.B.: *Stimulus Dependent Assembly Formation of Oscillatory Responses: I. Synchronization.* Neural Computation 3, pp. 155-166 (1991)

KS96 Kock, G., Serbedzija, B.: *Simulation of Artificial Neural Networks.* Systems Analysis Modelling Simulation (SAMS), Vol. 27, pp. 15-59 (1996)

Kud99 Kudla, H.: *Roma caput mundi regit orbis frena rotundi.* Lexikon der lateinischen Zitate, München 1999, Beck’sche Reihe (1999)

KSE92 Kreimeier, B., Schöne, M., Eckmiller, R.: *Communication Load Reduction for Neural Network Implementations on Message Passing Multicomputers.* Artificial Neural Networks (ICANN‘92), Elsevier (Amsterdam), vol. 2, pp. 1655-1658 (1992)

KSS93 Kreimeier, B., Schöne, M., Steiner, R., Eckmiller, R.: *How to Find a Near Optimal Mapping of Neural Networks onto Message Passing Multicomputers.* Proceedings of the ICANN‘93, Amsterdam, Springer (Heidelberg), pp. 309-312 (1993)

LS91 Levine, M.W., Shefner, J.M.: *Fundamentals of Sensation Perception.* Brooks/ Cole Publishing Company, California (1991)

Lu00 Lucent Technologies: *DSP16410 Data Sheet July 2000.* www.lucent.com (2000)

Maa97 Maass, W.: *Networks of Spiking Neurons: The Third Generation of Neural Network Models.* Neural Networks, Vol. 10, No. 9, pp. 1659-1671 (1997)

May97 Mayer-Lindenberg, F.: *A Heterogenous Parallel System Employing a Configurable Interconnection Network.* Proc. of the PDSC‘97, Washington (1997)

May98 Mayer-Lindenberg, F.: *A Universal Architecture for Parallel Embedded Systems.* Proc. of the PDPTA‘98, Las Vegas (1998)

May99 Mayer-Lindenberg, F.: *Passive Crossbar Interconnection for Parallel DSP Systems.* Proc. of the SHARC‘99, Hatfield (1999)

Mar80 Marcelja, S.: *Mathematical Description of the Responses of Simple Cortical Cells.* Journal Optical Society of America, Vol. 70, pp. 1297-1300 (1998)

MB98 Maass, W., Bishop, C.M.: *Pulsed Neural Networks.* A Bradford Book, MIT Press (1998)

MDP96 Monien, B., Diekmann, R., Preis, R.: *Lastverteilungsverfahren für Parallelrechner mit verteiltem Speicher.* Tagungsband der Sommerschule über part. DGL, KFA-Jülich, (1996)

MGG95 Müller, U., Gunzinger, A., Guggenbühl, W.: *Fast Neural Net Simulation with a DSP Processor Array.* IEEE Transactions on Neural Networks, Vol. 6, No. 1, pp. 203-213 (1995)

Mis97 Misra, M.: *Parallel Environments for Implementing Neural Networks.* Neural Computing Surveys, Vol. 1, pp. 48-60 (1997)

Möl95 Möller, A.: *Leistungsvergleich verschiedener Neuronenmodelle bei grundlegenden Aufgaben der visuellen Informationsverarbeitung*. Diplomarbeit, AG Neurophysik / FB Physik, Philipps-Universität Marburg / Lahn (1995)

Möl96 Möller, A.: *MNET (Marburger Neuronale Netze Beschreibungssprache) Version 2.0A.* Interne Dokumentation AG Neurophysik / FB Physik , Philipps-Universität Marburg / Lahn (1996)

MP43 McCulloch, W.S., Pitts, W.: *A Logical Calculus of the Ideas Immanent in Nervous Activity.* Bulletin of Mathematical Biophysics, 5, pp. 115-133 (1943)

MSP97 Mohraz, K., Schott, U., Pauly, M.: *Parallel Simulation of Pulse-Coded Neural Networks.* In: Proceedings of the IMACS World Congress ’97, Berlin, Vol. 6, pp. 523-528 (1997)

MS00 Mitsubishi Semiconductor: *M32R/D Product Information.* www.mitsubishi-electronics.com (2000)

Nau63 Naur, P.: *Revised Report on the Algorithmic Language ALGOL60.* Comm. of the ACM 6, 1, pp. 1-17 (1963)

NB94 Nierbur, E., Brettle, D.: *Efficient Simulation of Biological Neural Networks on Massively Parallel Supercomputers with Hypercube Architecture.* Advances in Neural Information Processing Systems 6, pp 904-910 (1994).

NS92 Nordström, T., Svensson, B.: *Using and Designing Massivly Parallel Computers for Artificial Neural Networks.* Journal of Parallel and Distributed Computing, 14, pp. 260-285 (1992)

PD97 Preis, R., Diekmann, R.: *PARTY - a software library for graph partitioning*. In: B.H.V. Topping, editor, Advances in Computational Mechanics with Parallel and Distributed Processing, pp 63-71 (1997)

Pla72 Playboy Magazine: *Playboy's Playmate of the Month - Miss November*. Playboy Enterprises Inc., vol 11/72, Centerfold (1972)

PPD89 Petrowski, A., Personnaz, L., Dreyfus, G., Girault, C.: *Parallel Implementations of Neural Network Simulations.* Hypercube and Distributed Computers, F. Andre and J.P. Verjus (Editors), Elsevier Science (North Holland), pp. 205-218 (1989)

Pre97-00 Preis, R.: Persönliche Kommunikation, Universität-GH Paderborn (1997-2000)

Pre00 Preis, R.: *Analyses and Design of Efficient Graph Partitioning Methods.* Dissertation, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (2000)

Rah99 Rahne, H.: *Parallelisierung der Simulation pulscodierter neuronaler Netze mittels digitaler Signalprozessoren.* Studienarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (1999)

Rah00 Rahne, H.: *Entwicklung und Implementierung eines Softwaresystems zur PCNN-Simulation auf dem Analog Devices ADSP21060.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

Ram92 Ramacher, U.: *SYNAPSE- A Neurocomputer That Synthesizes Neural Algorithms on a Parallel Systolic Engine.* Journal of Parallel and Distributed Computing, 14, pp. 306-318 (1992)

RD99 Rotter, S., Diesmann, M.: *Exact Digital Simulation of Time-Invariant Linear Systems with Application to Neuronal Modeling.* Biological Cybernetics, Vol. 81, pp. 381-402 (1999)

REJ97 Roth, U., Eckhardt, F., Jahnke, A., Klar, H.: *Efficient On-Line Computation of Connectivity: Architecture of the Connection Unit of NESPINN.* In: Proceedings of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary & Fuzzy Systems, Dresden, pp. 31-38 (1997)

Res93a Reski, T.: *Distributed Simulation of Synchronous Neural Networks.* Fifth Workshop on Neural Networks at The 1993 International Simulation Technology Conference, Society for Computer Simulation, pp. 276-281 (1993)

Res93b Reski, T.: *Simulation of Neural Networks on Massively Parallel Systems.* European Simulation Symposium, Verbraeck and Kerkhoffs (Eds.), Society for Computer Simulation International, pp. 767-773 (1993)

Res93c Reski, T.: *Mapping and Parallel Simulation of Synchronous Neural Networks on Multiprocessors.* Proceedings of the Euromicro Conference, pp. 681-688, Spain (1993)

Res99 Reski, T.: *Mapping and Parallel, Distributed Simulation of Neural Networks on Message Passing Multiprocessors.* Dissertation, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (1999)

RGF93 Ritz, R., Gerstner, W., Fuentes, U., v. Hemmen, J. L.: *A biologically motivated and analytically soluble model of collective oscillations in the cortex. II. Application to binding and pattern segmentation*, Biological Cybernetics 70, pp. 347-358 (1993)

RHG96 Rossmann, M., Hesse, B., Goser, K., Bühlmeier, A., Manteuffel, G.: *Implementation of a Biologically Inspired Neuron Model in FPGA.* Proc. of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MicroNeuro '96, IEEE Computer Society Press, pp. 322-329 (1996)

RHH90 Richert, P., Hess, G., Hosticka, B., Kesper, M., Schwarz, M.: *Distributed Processing Hardware for Realization of Artificial Neural Networks.* Parallel Processing in Neural Systems and Computers, Eckmiller, Hartmann and Hauske (Eds.), Elsevier (North Holland), pp.311-314 (1990)

RJK95 Roth, U., Jahnke, A., Klar, H.: *Hardware Requirements for Spike-Processing Neural Networks.* IWANN 95, Malaga, Spain, pp. 720-727 (1995)

RJK97 Roth, U., Jahnke, A., Klar, H.: *On-Line Hebbian Learning for Spiking Neurons: Architecture of the Weight-Unit of NESPINN.* Artificial Neural Networks ICANN‘97, pp. 1217-1222 (1997)

Rub98 Rubner, J.: *Tassen im Trommelfeuer.* Süddeutsche Zeitung Nr. 65, Beilage Umwelt-Wissenschaft-Technik, Donnerstag, 19. März 1998, pp. 1 (1998)

Rüc96 Rüchel, S.: *Design und Implementierung eines verteilten, objektorientierten Simulators zur Leistungsbewertung von Rechnernetzen.* Diplomarbeit, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (1996)

Sal98-00 Salzwedel, K.A.: Persönliche Kommunikation, Universität-GH Paderborn (1998-2000)

SAM00 Schönauer, T., Attasoy, S., Mehrtash, N., Klar, H.: *Simulation of a Digital Neuro-Chip for Spiking Neural Networks.* International Joint Conference on Neural Networks, IJCNN'00, Como, Italy (2000)

Sch93 Schwarz, M.: *Ein massiv paralleles Rechnersystem für die Emulation künstlicher neuronaler Netze und genetischer Algorithmen mit Anwendung in der Bildmustererkennung.* Dissertation, Universität-GH Duisburg, Fortschrittberichte VDI, Reihe 10, Nr. 236, VDI-Verlag (1993)

Sch95 Schmidt, F.: *Neuro- und Sinnesphysiologie.* Springer Verlag, Heidelberg (1995)

Sch95-00 Schäfer, M.: Persönliche Kommunikation, Universität-GH Paderborn (1995-2000)

Sch00 Schäfer, M.: *Lernen in Neurocomputern für große pulscodierte neuronale Netze.* Dissertation, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn, Paderborn (2000)

SER97 Stoecker, M., Eckhorn, R., Reitböck, H.J.: *Size and Position Invariant Visual Representation Supports Retinotopic Maps via Selective Backward Paths: A Dynamic Second Order Neural Network Model for a Possible Functional Role of Recurrent Connections in the Visual Cortex.* Neurocomputing, Vol. 17, pp. 111-132 (1997)

SGS96 Strey, A., Gutzmann, M.M., Stoyan, R.L.: *Effiziente parallele Implementierung neuronaler Netze.* 15. PARS-Workshop Programmiersprachen und hardwarenahe Programmierung, PARS Mitteilungen Nr. 15, pp. 137-148 (1996)

SH99 Schäfer, M., Hartmann, G.: *A Flexible Hardware Architecture for Online Hebbian Learning in the Sender-Oriented Neurocomputer Spike 128K.* In: Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, Granada, Spain, pp. 316-323 (1999)

SHK94 Schwarz, M., Hosticka, B.J., Kesper, M., Richert, P., Scholles, M.: *A Parallel DSP-Based Neural Network Emulator with CMOS VLSI Packet Switching Hardware.* Proc. of the International Conference on Application Specific Array Processors, pp.382-391 (1994)

Sil00 Silva-Fernandes, R.-A.: *Ein Testsystem für Signalprozessoren in der parallelen PCNN-Simulation.* Studienarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

SJR98 Schönauer, T., Jahnke, A., Roth, U., Klar, H.: *Digital Neurohardware: Principles and Perspectives.* In: Proc. Neuronale Netze in der Anwendung - NN‘98, Magdeburg, pp. 101-106, (1998)

SK91 Schillen, T.B., König, A.: *Stimulus Dependent Assembly Formation of Oscillatory Responses: II. Desynchronization.* Neural Computation 3, pp. 167-178 (1991)

SMJ98 Schönauer, T., Mehrtash, N., Jahnke, A., Klar, H.: *MASPINN: Novel Concepts for a Neuro-Accelerator for Spiking Neural Networks.* In: Workshop on Virtual Intelligence and Dynamic Neural Networks - VIDYNN’98, Stockholm, (1998)

SMK98 Schönauer, T., Mehrtash, N., Klar, H.: *Architecture of a Neuroprocessor Chip for Pulse-Coded Neural Networks.* Int. Conf. on Comp. Intelligence and Neuros. - ICCIN‘98, JCIS‘98, Research Triangle Park, NC (USA), pp. 17-20 (1998)

SPEC95 Standard Performance Evaluation Corporation: *SPEC 95 CPU Benchmark.* www.spec.org (1995)

SRE96 Stoecker, M., Reitböck, J.R., Eckhorn, R.: *A Neural Network for Scene Segmentation by Temporal Coding.* Neurocomputing, Vol. 11, pp. 123-134 (1996)

SSW00 Schäfer, M., Schönauer, T., Wolff, C.: *Simulation pulscodierter neuronaler Netze - Methoden und Hardwarearchitekturen.* Interner Bericht Nr. Spike 01/00, Grundlagen der Elektrotechnik, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

Str94 Strasdat, H.: *Parallele Simulation synchroner neuronaler Netze auf nachrichtengekoppelten Multiprozessorsystemen.* Diplomarbeit, Fachbereich 17 Mathematik-Informatik, Universität-GH Paderborn (1994)

Str99a Strey, A.: *EpsiloNN - A Tool for the Abstract Specification and Parallel Simulation of Neural Networks.* Systems Analysis Modelling Simulation (SAMS), Special Issue on Simulation of Artificial Neural Networks, Gordon&Breach (1999)

Str99b Strey, A.: *A unified model for the simulation of artificial and biology-oriented neural networks.* Engineering Applications of Bio-Inspired Artificial Neural Networks (Mira, J. and Sanchez-Andres, J., eds.), Lecture Notes in Computer Science 1607, Springer, Berlin, pp. 1-10 (1999)

Str00 Stracke, P.: *VME-Bus-Anbindung eines DSP-Parallelrechners mit Hilfe von FPGAs.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

SW97 Schott, U., Weitzel, L.: Persönliche Kommunikation, Philipps-Universität Marburg (1997)

Teu96 Teuner, A.: *Untersuchung und Anwendungen der Gabor-Transformation für die Bildkompression und Bildsegmentierung.* Dissertation, Universität-GH Duisburg, Fortschrittberichte VDI, Reihe 10, Nr. 403, VDI-Verlag (1996)

Thi99 Thiem, J.: Persönliche Kommunikation, Universität-GH Paderborn (1999)

TI00 Texas Instruments: *TMS320C6203 & TMS320C64x Technical Brief.* www.ti.com (2000)

TN92 Tay, O.N., Noakes, P.D.: *The Use of Cache Memory in Neurocomputer Design.* Proc. of ICNN‘92, Vol. II, pp. 571-576 (1992)

TWH00 Thiem, J., Wolff, C., Hartmann, G.: *Biology-Inspired Early Vision System for a Spike Processing Neurocomputer*. IEEE International Workshop on Biologically Motivated Computer Vision (BMCV2000), pp. 387-396, Seoul (Korea) (2000)

Ull00 Ullrich, C.: *Entwicklung eines Speicher-Controllers in FPGA-Technik für einen DSP-Parallelrechner.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

vdM81 von der Malsburg, C.: *The Correlation Theory of Brain Function.* Internal Report 81-2, Dept. of Neurobiology, Max-Planck-Institute for Biophysical Chemestry, Göttingen (1981)

Wan00 Wandtner, R.: *Aufbruch ins Gehirn*. Frankfurter Allgemeine Zeitung Nr. 83, Feuilleton, Freitag, 7.April 2000, pp. 53 (2000)

Wat94 Watts, L.: *Event Driven Simulation of Networks of Spiking Neurons.* Advances in Neural Information Processing Systems, vol. 6, pp. 927-934 (1994)

Weg00 Wegmann, G-J.: *Kommunikationseffiziente parallele Simulation großer PCNN auf Cluster-Computern.* Diplomarbeit, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (2000)

WKS97 Weitzel, L., Kopecz, K., Spengler, C., Eckhorn, R., Reitböck, H. J.: *Contour Segmentation with Recurrent Neural Networks of Pulse-Coding Neurons.* CAIP’97, Kiel (1997)

WHR99a Wolff, C., Hartmann, G., Rückert, U.: *ParSPIKE - A Parallel DSP-Accelerator for Dynamic Simulation of Large Spiking Neural Networks.* In: Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, Granada, Spain, pp.324-331 (1999)

WHR99b Wolff, C., Hartmann, G., Rahne, H.: *Parallele Simulation großer pulscodierter neuronaler Netze auf DSPs*. In: DSP Deutschland '99, Grundlagen, Tools, Applikationen, München, pp.267-273 (1999)

WSW98a Wegmann, G-J., Salzwedel, K.A., Wolff, C.: *Dokumentation der Netzbeschreibungssprache MNET 2.1 für den Spike128k.* Interner Bericht Nr. Spike 01/98, Grundlagen der Elektrotechnik, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (1998)

WSW98b Wegmann, G-J., Salzwedel, K.A., Wolff, C.: *Hilfsblatt zum Compilerbau mit Eli.* Interner Bericht Nr. Spike 02/98, Grundlagen der Elektrotechnik, Fachbereich 14 Elektrotechnik und Informationstechnik, Universität-GH Paderborn (1998)

WWK97 Walker, M., Wang, H., Kartamihardjo, S., Roth. U.: *SimSPINN - A Simulator for Spike-Processing Neural Networks.* Proceedings of IMACS‘97, (1997)

Zel94 Zell, A.: *Simulation neuronaler Netze.* Addison-Wesley (Deutschland), (1994)

Glossar

Abklingliste	Ereignisliste zum Sammeln der in der Abklingphase zu bearbeitenden Neuronen (siehe Kap. 3.4).
Aktionspotential	Bei Überschwelligkeit vom Neuron ausgesandtes Signal, auch Puls oder Spike genannt (siehe Kap. 2.1).
ANSI-C	Prozedural orientierte höhere Programmiersprache in ANSI-Norm.
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit. Speziell entwickelter und schaltungstechnisch realisierter Chip.
Axon	Fortsatz der Nervenzelle, über den das Aktionspotential, d.h. der Spike, an die Synapsen der Zelle und darüber an die Nachfolgezellen übertragen wird (siehe Kap. 2.1).
BSA	**B**lock-**S**tart-**A**dresse zum Auffinden eines EIBs (siehe Kap. 3.4).
BSS	**B**lock-**S**tart-**S**peicher für die BSAs (siehe Kap. 3.4).
Compiler	Programm zur Übersetzung aus einer Sprache, die auf einer kontextfreien Grammatik basiert, in eine andere Sprache.
Compiler-Compiler	Programm zur Erzeugung eines Compilers.
CU	Kommunikationseinheit (**C**ommunication **U**nit) des ParSPIKE.
Dendrit	Zellfortsatz des Neurons, an den über die Synapsen anderer Neuronen Signale übertragen werden. Dendriten bilden den Eingang des Neurons, an ihnen baut sich das postsynaptische Dendritenpotential auf.
DRAM	**D**ynamic **R**andom **A**ccess **M**emory
DSP	**D**igitaler **S**ignal**p**rozessor (siehe Kap. 6.2)
Echtzeitsimulation	Im Zusammenhang der Simulation von Bildverarbeitungs-PCNN die Simulation von einem Zeitschritt pro Millisekunde [DDW98].

EIB	**E**rregungs-**I**nformations-**B**lock zur Speicherung der Verbindungen eines Neurons (siehe Kap. 3.4).
ELI	Compiler-Compiler u.a. aus der Kastens-Gruppe [GHL92].
FIFO	Listenspeicher nach dem **F**irst **I**n **F**irst **O**ut Prinzip.
FPGA	**F**ield **P**rogrammable **G**ate **A**rray. Komplexer, beliebig oft reprogrammierbarer Logikbaustein auf Basis vorkonfigurierter Elemente.
Ganglien	Erste Zellschicht der Retina, die ihre Aktivität pulscodiert (siehe Kap. 2.1).
Linkverbindung	Im Bereich der Rechnertechnik eine Punkt-zu-Punkt-Verbindung mit geringer Wortbreite.
LU	Lerneinheit im ParSPIKE-Konzept (**L**earn **U**nit)
MASPINN	Spezialrechner für die PCNN-Simulation auf Basis eines ASICs (NeuroPipe-Chip) für die Neuronenberechnungen mit vier Prozessorpipelines (siehe Kap. 7.4.3).
Membranpotential	Aktivierungszustand eines Neurons. Überschreitet dieser Wert eine Schwelle, so feuert das Neuron einen Spike.
MNET	**M**arburger-Neuronale-**Net**ze-Sprache (siehe Kap. 5.2)
NA	**N**euronen**a**dresse
NESPINN	Vorgängerkonzept zu MASPINN
nrc	**n**on **r**egular **c**onnection. Verknüpfungsstrukturen, die sich nicht mittels einer EIB-Maske beschreiben lassen.
on-Chip-Speicher	Speicher, der auf dem Silizium-Die des Prozessorchips integriert ist.
ParSPIKE	Konzept einer parallelen Spezialhardware auf Basis von DSPs (siehe Kap. 6.1).
PCI	**P**eripheral **C**omponent **I**nterface, Bussystem für Steckkarten.
PCNN	**P**uls**c**odierte **n**euronale **N**etze
PE	Prozessorelement
Pipelining	Hintereinanderschaltung separater Verarbeitungseinheiten für die Teilschritte eines umfangreicheren Berechnungsschrittes.
postsynaptisch	Signalempfangender Teil bezüglich der Spikeübertragung.

PostSpike	Spike nach der Vervielfältigung bezüglich der Netztopologie. Repräsentiert durch die Neuronenadresse des Empfängerneurons.
präsynaptisch	Signalsendender Teil bezüglich der Spikeübertragung.
PreSpike	Spike vor der Vervielfältigung bezüglich der Netztopologie. Repräsentiert durch die Neuronenadresse des Senderneurons.
PVM	**P**arallele **V**irtuelle **M**aschine (siehe Kap. 5.1)
rc	**r**egular **c**onnection. Verknüpfungsstrukturen, die sich mittels einer EIB-Maske beschreiben lassen.
Refraktärzeit	Zeit nach der Spikeemission, in der das Neuron nicht erneut feuern kann (siehe Kap. 2.1).
Retina	Zellverband im hinteren Teil des Auges, der ein Bild in neuronale Aktivität umsetzt (siehe Kap. 2.1).
rezeptives Feld	Bildbereich, auf den ein Neuron der Sehbahn reagiert (siehe Kap. 2.1).
Routing	Datenübertragung aufgrund von Zielinformationen in den Daten durch autonome Einheiten.
SDRAM	**S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory
SPIKE128k	Neurocomputer für die PCNN-Simulation (siehe Kap. 3.5)
Spikeliste	Ereignisliste zum Sammeln der Spikes (siehe Kap. 3.4).
SRAM	**S**tatic **R**andom **A**ccess **M**emory
Switching	Datenübertragung durch Schalten fester Übertragungswege.
Synapse	Endknöpfchen am Axon zur Spikeübertragung an Nachfolgeneuronen (siehe Kap. 2.1).
WMC	**W**eight **M**emory **C**ontroller (siehe Kap. 6.4)
VHDL	Beschreibungssprache für digitale Schaltungen.
VME	**V**ersa **M**odule **E**urocard, Bussystem für Steckkarten.

Anhang A - Herleitung der Simulationsgleichungen

Das Eckhorn Neuronenmodell ist eine heuristische Nachbildung der zeitlichen Charakteristik der Spike-Emission einer biologischen Nervenzelle. Es ist daher nicht möglich, das Modell oder die Modellparameter zu messen oder aus den Messungen zu bestimmen. Vielmehr wird eine Modellierung gewählt, die vermuten läßt, daß sich mit ihrer Hilfe die zeitlichen Effekte der Spike-Emission nachbilden lassen. Die Wahl von linearen Differentialgleichungen erster Ordnung für die dendritischen Potentiale und die dynamische Schwelle erfolgt daher im wesentlichen aufgrund der Tatsache, daß sich damit die denkbar einfachste Möglichkeit zur Beschreibung des Anregungs- und Abklingvorganges ergibt. Die Umsetzung in iterativ zu berechnende Simulationsgleichungen in Kap. 3.2 kann daher prinzipiell auch heuristisch erfolgen, da bekannt ist, daß sich das gewünschte Verhalten mit Hilfe von Leckintegratoren erreichen läßt. Durch eine geeignete Wahl der Parameter läßt sich dann das im biologischen Vorbild beobachtete Spikeverhalten auch in der Simulation erreichen.

Trotz dieser Argumente, die eine konsistente Herleitung der Simulationsgleichungen aus einem kontinuierlichen Modell überflüssig erscheinen lassen, soll gerade eine solche Herleitung hier skizziert werden, da sich die Betrachtung einer kontinuierlichen Modellierung als hilfreich zu Analysezwecken gezeigt hat [Thi99] und der Transfer der so gewonnenen Erkenntnisse in die Simulation durch einen konsistenten Übergang erleichtert wird.

Dazu soll zuerst exemplarisch die Herleitung der Simulationsgleichungen für das Linkingpotential LP aus der Differentialgleichungsbeschreibung in Kap. 2.3.3 gezeigt werden (siehe auch [Ibe99][Thi99]). Dort wurde der Aktivierungszustand $a_{i,LP}(t)$ des Linkingpotentials durch den Zusammenhang

$$\tau_{LP} \cdot \frac{da_{i,LP}(t)}{dt} = net_{i,LP}(t) - a_{i,LP}(t) \qquad \text{(Gl. 9.1)}$$

beschrieben, wobei sich die Netzeingabe $net_{i,LP}(t)$ mit

$$net_{i,LP}(t) = \sum_j w_{ij} \cdot o_j(t) \qquad \text{(Gl. 9.2)}$$

aus der Ausgabe $o_j(t)$ der vorangegangenen Neuronen und dem Gewicht w_{ij} der entsprechenden Verbindungen zusammensetzt. Durch Anwendung der einseitigen Laplace-Transformation ergibt sich als Lösung die Faltung

$$a_{i,\,LP}(t) = h_{LP}(t) * net_{i,\,LP}(t) \qquad \text{(Gl. 9.3)}$$

mit der Impulsantwort

$$h_{LP}(t) = \frac{1}{\tau_{LP}} \cdot \sigma(t) \cdot e^{-\frac{t}{\tau_{LP}}} \qquad \text{(Gl. 9.4)}$$

wobei $\sigma(t)$ die Sprungfunktion ist.

$$\sigma(t) = \begin{cases} 1 \text{ , falls} & t > 0 \\ 0 \text{ , falls} & t \le 0 \end{cases} \qquad \text{(Gl. 9.5)}$$

Zur Diskretisierung wird die Lösung zu den Zeitpunkten $t=nT$ mit $n \in [0 \ldots N]$ bestimmt, wobei T die Zeitschrittlänge und N die Anzahl der zu simulierenden Zeitschritte ist. Diese Lösung wird mit $\tilde{LP}_i(nT)$ bezeichnet. Das Abtasttheorem sei im weiteren stets erfüllt.

$$\begin{aligned} \tilde{LP}_i(nT) &= a_{i,\,LP}(t = nT) \\ \tilde{LP}_i(nT) &= net_{i,\,LP}(nT) * \frac{1}{\tau_{LP}} \cdot e^{-\frac{nT}{\tau_{LP}}} \cdot \sigma(nT) \end{aligned} \qquad \text{(Gl. 9.6)}$$

Die Faltung läßt durch eine Faltungssumme

$$\tilde{LP}_i(nT) \approx \sum_{\kappa\,=\,-\infty}^{\infty} net_{i,\,LP}(\kappa T) \cdot h_{D,\,LP}((n-\kappa)T) \qquad \text{(Gl. 9.7)}$$

annähern. Um diese Annäherung vornehmen zu können, muß die Impulsantwort des kontinuierlichen Systems diskretisiert werden. Im weiteren wird gezeigt, daß für eine solche diskrete Impulsantwort der Zusammenhang

$$h_{D,\,LP}(nT) \approx T \cdot h_{LP}(t = nT) \qquad \text{(Gl. 9.8)}$$

gilt, d.h. die Zeitschrittlänge T taucht als Proportionalitätsfaktor auf.

Ziel der diskreten PCNN-Simulation ist es, durch Simulation genau die Ausgangsgrößen $y(nT)$ zu gewinnen, die in einer Messung zu den Zeitpunkten nT am biologischen Vorbild auftreten. Dazu werden wiederum in das diskrete System die Eingangsgrößen des biologi-

schen Vorbildes zu den Zeitpunkten *nT* eingespeist. Dieses Vorgehen führt dazu, daß zur diskreten Simulation die digitale Umsetzung der Daten vor das zu simulierende System verlegt wird, d.h. die Impulsantwort des Systems muß diskretisiert werden. Dieses läßt sich einfach erreichen, indem die Signalübertragung für beide Fälle dargestellt wird.

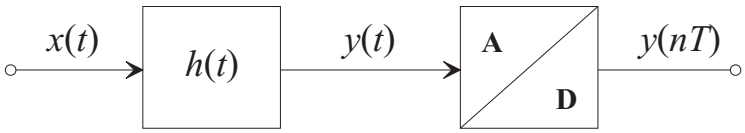

Abb. 9-1 Diskretisierung der Ausgänge des kontinuierlichen Systems

Der kontinuierliche Ausgang *y*(*t*) ergibt sich zu

$$y(t) = h(t) * x(t) = \int_{-\infty}^{t} h(t-\tau)x(\tau)d\tau \qquad \text{(Gl. 9.9)}$$

und die Werte zu den diskreten Zeitpunkten *nT* ergeben sich entsprechend zu

$$y(nT) = \int_{-\infty}^{nT} h(t-\tau)x(\tau)d\tau \approx \sum_{\kappa=-\infty}^{n} T \cdot h(nT-\kappa T)x(\kappa T). \qquad \text{(Gl. 9.10)}$$

Im Fall der diskreten Simulation stehen die Eingangswerte des biologischen Vorbildes zu den Zeitpunkten *nT* als Eingangsdaten *x*(*nT*) zur Verfügung, so daß zur Erzeugung der Ausgangswerte *y*(*nT*) nur eine diskrete Faltungssumme berechnet werden muß.

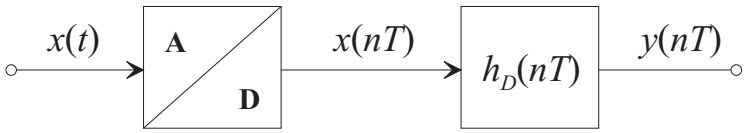

Abb. 9-2 Verarbeitung diskreter Eingangswerte durch die diskrete Simulation

Da die Ausgangswerte *y*(*nT*) für beide dargestellten Fälle gleich sein sollen, ergibt sich aus der diskreten Faltung mit

$$y(nT) = h_D(nT) * x(nT) = \sum_{\kappa=-\infty}^{n} h_D(nT-\kappa T)x(\kappa T) \qquad \text{(Gl. 9.11)}$$

die Forderung für $h_D(nT)$, daß

$$h_D(nT) \approx T \cdot h(t = nT) \qquad \text{(Gl. 9.12)}$$

gilt (Gl. 9.8).Nach diesem Exkurs kann nun die weitere Herleitung des Dendritenpotentials nach (Gl. 9.7) erfolgen. Das Dendritenpotential wird nunmehr als eine solche Faltungssumme angenommen und ergibt sich für die weitere Betrachtung zur folgenden Darstellung.

$$\tilde{LP}_i(nT) = T \cdot \sum_{\kappa = -\infty}^{\infty} net_{i,\,LP}(\kappa T) \cdot \frac{1}{\tau_{LP}} \cdot e^{-\frac{(n-\kappa)T}{\tau_{LP}}} \cdot \sigma((n-\kappa)T) \qquad \text{(Gl. 9.13)}$$

Mit den Annahmen, daß $net_{i,\,LP}(nT) = 0$ für $n < 0$ und $\sigma((n-\kappa)T) = 0$ für $\kappa > n$ gelten, läßt sich die folgende Vereinfachung treffen.

$$\tilde{LP}_i(nT) = T \cdot \sum_{\kappa = 0}^{n} net_{i,\,LP}(\kappa T) \cdot \frac{1}{\tau_{LP}} \cdot e^{-\frac{(n-\kappa)T}{\tau_{LP}}} \qquad \text{(Gl. 9.14)}$$

Im Rahmen einer Zeitschrittsimulation soll diese Summe iterativ aus den Netzeingaben und den Werten der Summe im vorangegangenen Zeitschritt berechnet werden. Dazu wird zunächst die Summe zum Zeitpunkt $t = (n\text{-}1)T$ berechnet.

$$\tilde{LP}_i((n-1)T) = \sum_{\kappa = 0}^{n-1} net_{i,\,LP}(\kappa T) \cdot \frac{T}{\tau_{LP}} \cdot e^{-\frac{((n-1)-\kappa)T}{\tau_{LP}}} \qquad \text{(Gl. 9.15)}$$

Durch einige Umformungen ergibt sich daraus für den Zeitpunkt nT die folgende Rekursion.

$$\begin{aligned}
\tilde{LP}_i((n-1)T) &= \sum_{\kappa = 0}^{n} net_{i,\,LP}(\kappa T) \cdot \frac{T}{\tau_{LP}} \cdot e^{-\frac{((n-1)-\kappa)T}{\tau_{LP}}} - net_{i,\,LP}(nT) \cdot \frac{T}{\tau_{LP}} \cdot e^{\frac{T}{\tau_{LP}}} \\
&= e^{\frac{T}{\tau_{LP}}} \cdot \sum_{\kappa = 0}^{n} net_{i,\,LP}(\kappa T) \cdot \frac{T}{\tau_{LP}} \cdot e^{-\frac{(n-\kappa)T}{\tau_{LP}}} - net_{i,\,LP}(nT) \cdot \frac{T}{\tau_{LP}} \cdot e^{\frac{T}{\tau_{LP}}} \\
\Rightarrow \sum_{\kappa = 0}^{n} net_{i,\,LP}(\kappa T) \cdot \frac{T}{\tau_{LP}} \cdot e^{-\frac{(n-\kappa)T}{\tau_{LP}}} &= \tilde{LP}_i((n-1)T) e^{-\frac{T}{\tau_{LP}}} + \frac{T}{\tau_{LP}} \cdot net_{i,\,LP}(nT) \\
\Rightarrow \tilde{LP}_i(nT) &= \tilde{LP}_i((n-1)T) e^{-\frac{T}{\tau_{LP}}} + \frac{T}{\tau_{LP}} \cdot net_{i,\,LP}(nT)
\end{aligned} \qquad \text{(Gl. 9.16)}$$

Diese Darstellung entspricht (Gl. 3.4) und läßt sich bezüglich der Abklingphase und der Erregungsphase aufteilen, sowie durch die Substitution

$$\tilde{LP}_i(nT) = LP_i(nT) - LP_{i,min} \qquad \text{(Gl. 9.17)}$$

in die Darstellung in (Gl. 3.6) bringen. Zudem kann zur vereinfachten Handhabung

$$\tilde{net}_{i,LP}(nT) = \frac{T}{\tau_{LP}} \cdot net_{i,LP}(nT) \qquad \text{(Gl. 9.18)}$$

substituiert werden. Die Substitution impliziert eine Modifikation der Gewichte, indem mit

$$\tilde{net}_{i,LP}(nT) = \frac{T}{\tau_{LP}} \cdot \sum_j w_{ij} \cdot o_j(nT) = \sum_j \tilde{w}_{ij} \cdot o_j(nT) \qquad \text{(Gl. 9.19)}$$

ein modifiziertes Gewicht

$$\tilde{w}_{ij} = \frac{T}{\tau_{LP}} w_{ij} \qquad \text{(Gl. 9.20)}$$

eingeführt wird. Diese Modifikation der Gewichte ist prinzipiell unerheblich, da synaptische Gewichte nicht im biologischen Vorbild gemessen werden können und der absolute Gewichtswert daher im Rahmen der Modellierung heuristisch festgelegt wird. Die Modifikation ist jedoch notwendig, da die Integrationszeit des Leckintegrators τ_{LP} unabhängig von der Zeitschrittlänge *T* aus dem kontinuierlichen Modell übernommen wird und dementsprechend eine Anpassung der Gewichte erfolgen muß, um zu gleichen Dendritenpotentialen zu kommen. Diese Anpassung muß daher bei der Übernahme von Gewichten aus dem kontinuierlichen Modell berücksichtigt werden.

Für die weiteren Teilpotentiale EP_1, EP_2 und *IP* erfolgt die Herleitung analog.

Die Herleitung der Simulationsgleichung für die dynamische Schwelle $DS_i(nT)$ wird ähnlich zum Linkingpotential durchgeführt. Mit $\Delta\tilde{DS}_i = \Delta\theta_i$ folgt aus der Faltung (Gl. 2.23):

$$\begin{aligned} \tilde{DS}_i(nT) &= \theta_i(t = nT) \\ \tilde{DS}_i(nT) &= \Delta\tilde{DS}_i \cdot \frac{1}{\tau_{DS}} \cdot e^{-\frac{nT}{\tau_{DS}}} \cdot \sigma(nT) * o_i(nT) \end{aligned} \qquad \text{(Gl. 9.21)}$$

Aus dieser Darstellung läßt sich entsprechend zu den Umformungen des Linkingpotentials eine diskrete Faltungssumme ermitteln, die analog zu (Gl. 9.14) die Form

$$\tilde{DS}_i(nT) = \Delta\tilde{DS}_i \cdot \frac{T}{\tau_{DS}} \cdot \sum_{\kappa=0}^{n} e^{-\frac{(n-\kappa)T}{\tau_{DS}}} \cdot o_i(\kappa T) \qquad \text{(Gl. 9.22)}$$

ergibt. Dabei wird analog zu (Gl. 9.20) für $\Delta\tilde{DS}_i$ die Darstellung

$$\Delta DS_i = \frac{T}{\tau_{DS}}\Delta\tilde{DS}_i \qquad \text{(Gl. 9.23)}$$

substituiert. Zur iterativen Berechnung wird wie in (Gl. 9.15) zunächst die Summe zum Zeitpunkt $t = (n\text{-}1)T$ berechnet, um daraus den Ausdruck für $t = nT$ zu gewinnen.

$$\begin{aligned}
\tilde{DS}_i((n-1)T) &= \Delta DS_i \cdot \sum_{\kappa=0}^{n-1} e^{-\frac{((n-1)-\kappa)T}{\tau_{DS}}} \cdot o_i(\kappa T) \\
\tilde{DS}_i((n-1)T) &= \Delta DS_i \cdot \left(\sum_{\kappa=0}^{n} e^{-\frac{((n-1)-\kappa)T}{\tau_{DS}}} \cdot o_i(\kappa T) - e^{\frac{T}{\tau_{DS}}} \cdot o_i(nT) \right) \\
\tilde{DS}_i((n-1)T) &= \Delta DS_i \cdot \left(\sum_{\kappa=0}^{n} e^{-\frac{(n-\kappa)T}{\tau_{DS}}} \cdot e^{\frac{T}{\tau_{DS}}} \cdot o_i(\kappa T) - e^{\frac{T}{\tau_{DS}}} \cdot o_i(nT) \right) \qquad \text{(Gl. 9.24)} \\
\tilde{DS}_i((n-1)T) &= \Delta DS_i \cdot e^{\frac{T}{\tau_{DS}}} \cdot \left(\sum_{\kappa=0}^{n} e^{-\frac{(n-\kappa)T}{\tau_{DS}}} \cdot o_i(\kappa T) - o_i(nT) \right) \\
\Rightarrow \Delta DS_i \cdot \sum_{\kappa=0}^{n} e^{-\frac{(n-\kappa)T}{\tau_{DS}}} \cdot o_i(\kappa T) &= \tilde{DS}_i((n-1)T) \cdot e^{-\frac{T}{\tau_{DS}}} + \Delta DS_i \cdot o_i(nT)
\end{aligned}$$

Dieser entspricht wiederum $\tilde{DS}_i(nT)$.

Aus der Ersetzung

$$\tilde{DS}_i(nT) = DS_i(nT) - DS_{i,\,min} \qquad \text{(Gl. 9.25)}$$

ergibt sich auch hier die Darstellung der (Gl. 3.9).

Anhang B - Simulationsergebnisse

Nachfolgend sind zusätzlich zu den Angaben in Kap. 5 einige weitere Verläufe zur Lastverteilung in der Abklingphase (a) und in der Erregungsphase (e) sowie zur Kommunikationslast auf Basis der Cachetrefferzahl (h_1) aufgeführt. Den Verläufen sind die Kennungen der jeweiligen Partitionierungsstrategien zugeordnet (siehe Kap. 5.2).

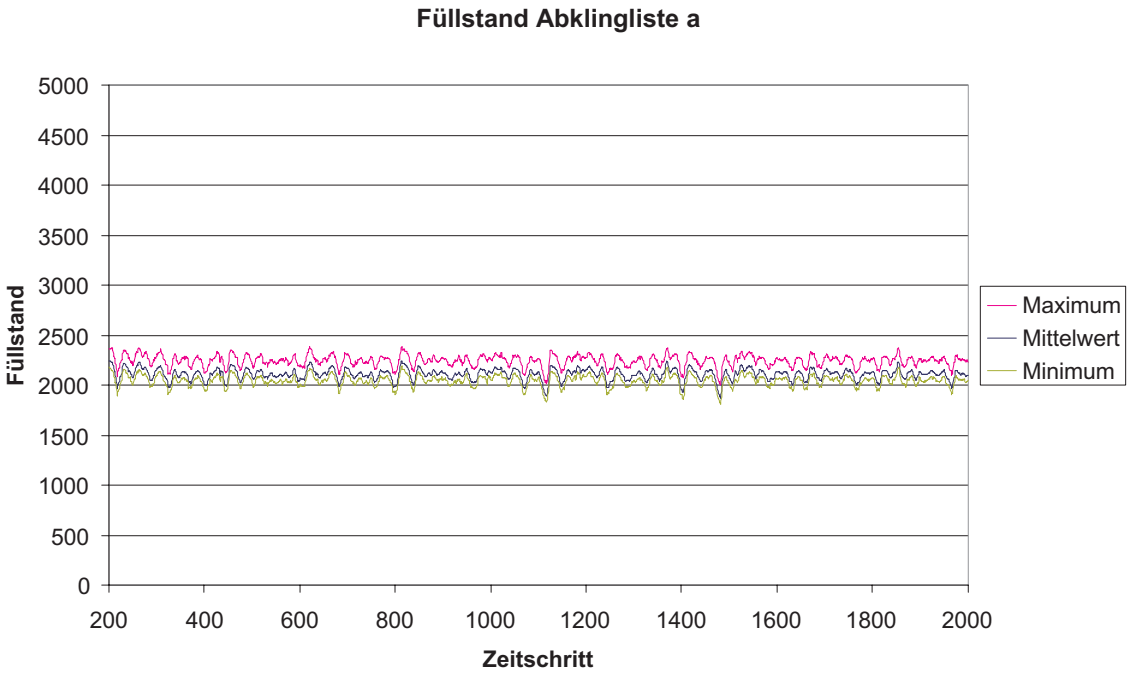

Abb. 10-1 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei Verteilung einzelner Neuronen mittels des einfachen Modulo-Verfahrens (1)

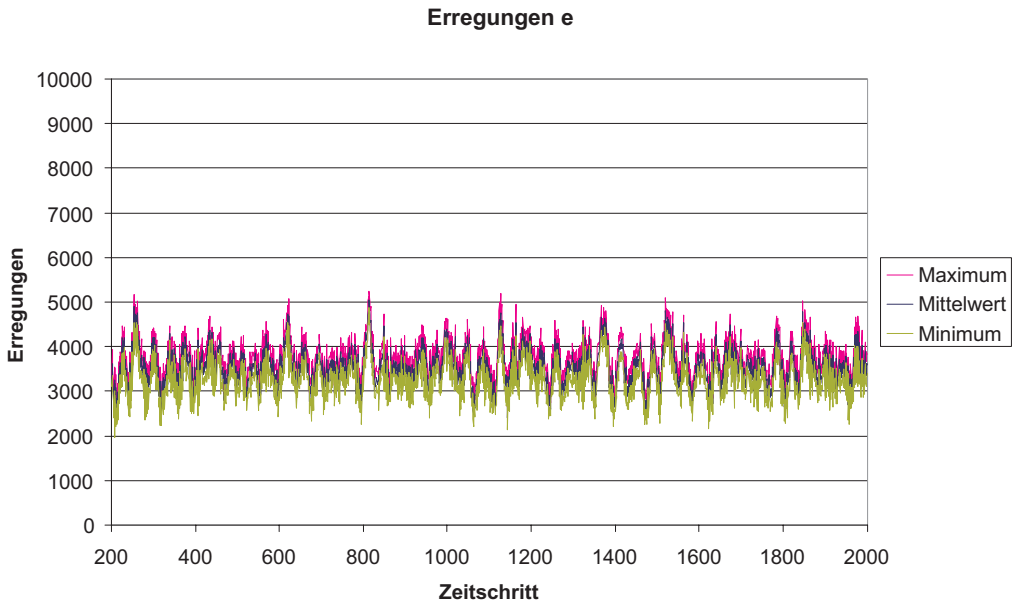

Abb. 10-2 Lastverteilung in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei Verteilung einzelner Neuronen mittels des einfachen Modulo-Verfahrens (1)

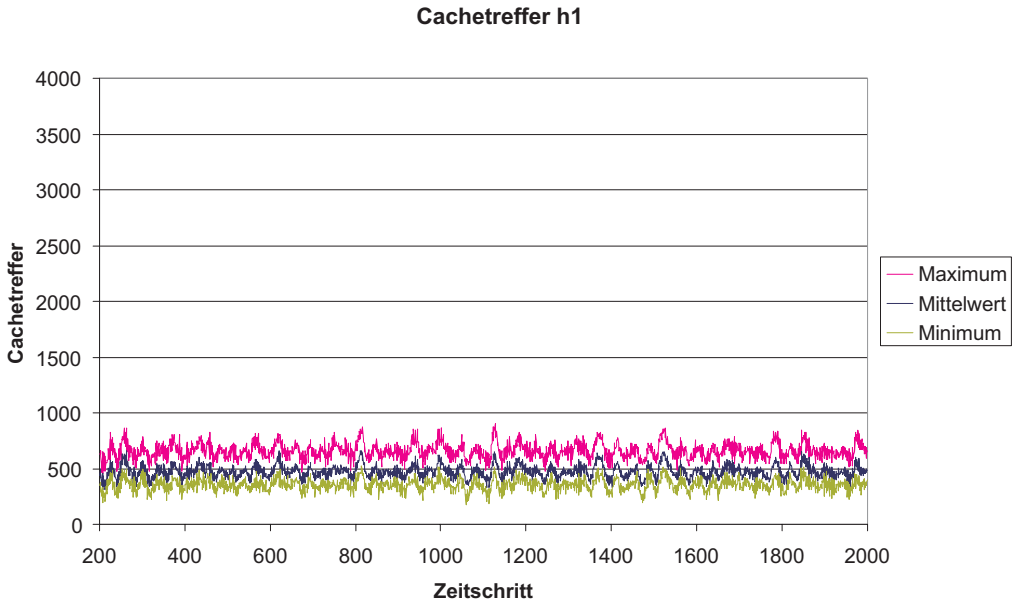

Abb. 10-3 Cachetreffer in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei Verteilung einzelner Neuronen mittels des einfachen Modulo-Verfahrens (1)

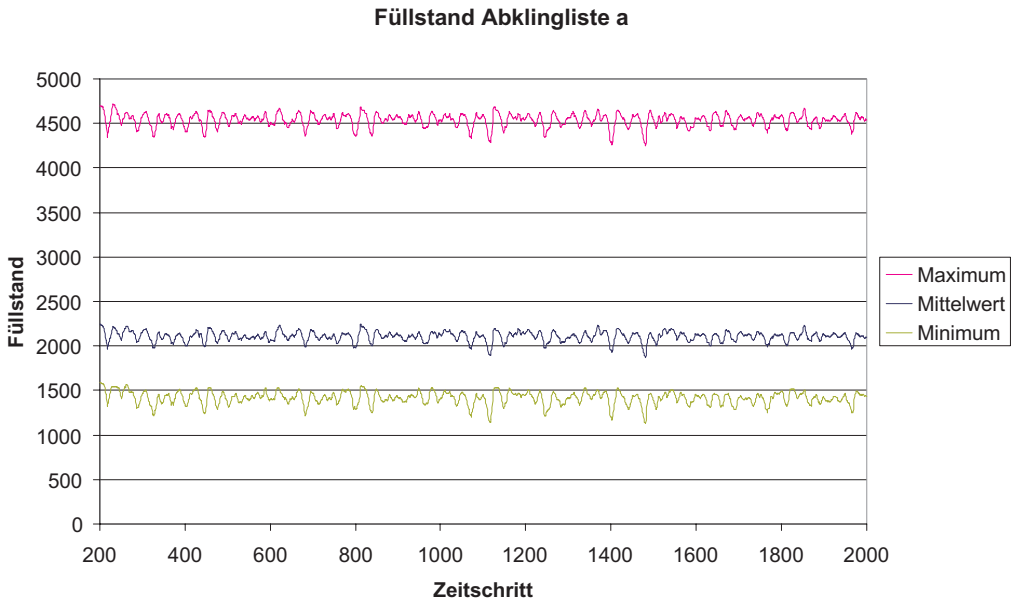

Abb. 10-4 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei Verteilung ganzer Neuronenlagen mittels des einfachen Modulo-Verfahrens (3)

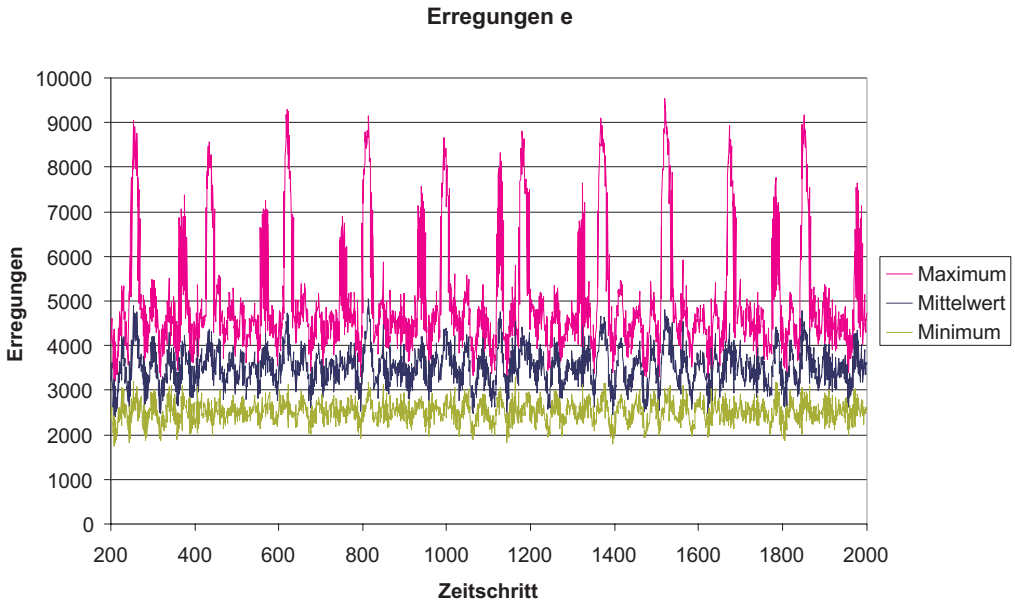

Abb. 10-5 Lastverteilung in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei Verteilung ganzer Neuronenlagen mittels des einfachen Modulo-Verfahrens (3)

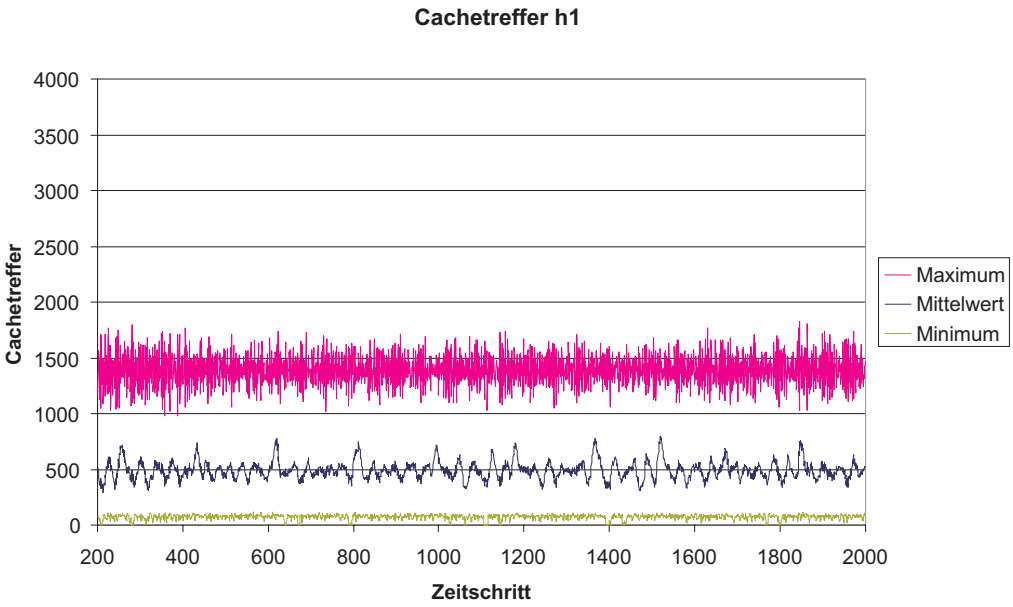

Abb. 10-6 Cachetreffer in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei Verteilung ganzer Neuronenlagen mittels des einfachen Modulo-Verfahrens (3)

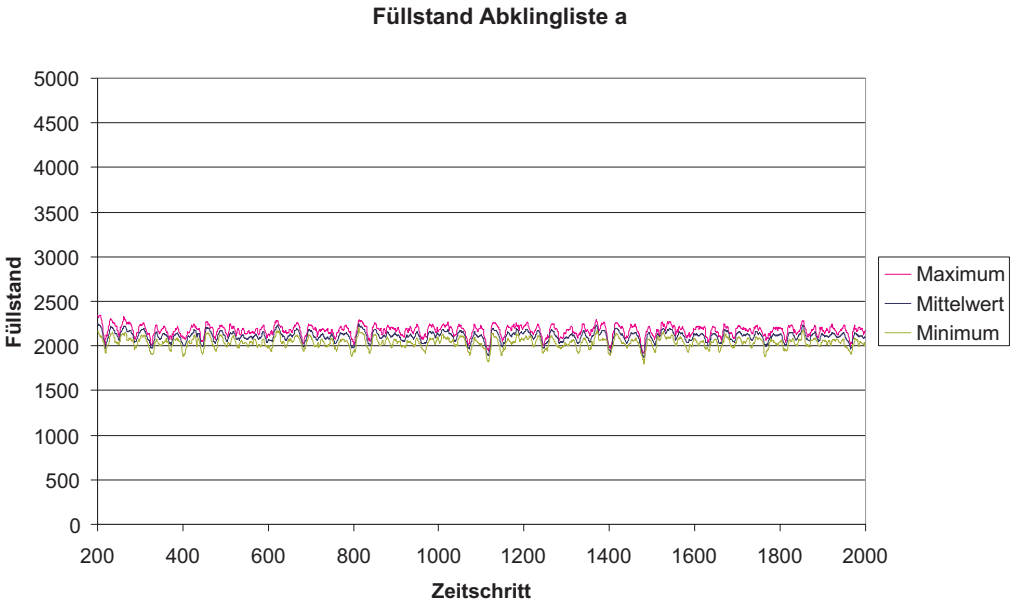

Abb. 10-7 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer Kombination der Breitensuche mit der Säulenpartitionierung (5)

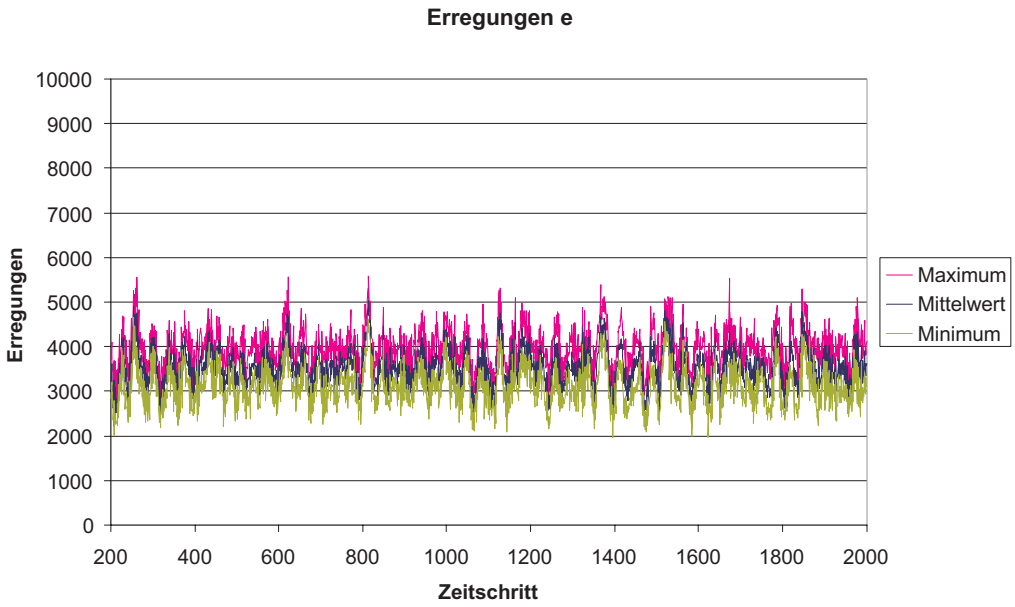

Abb. 10-8 Lastverteilung in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer Kombination der Breitensuche mit der Säulenpartitionierung (5)

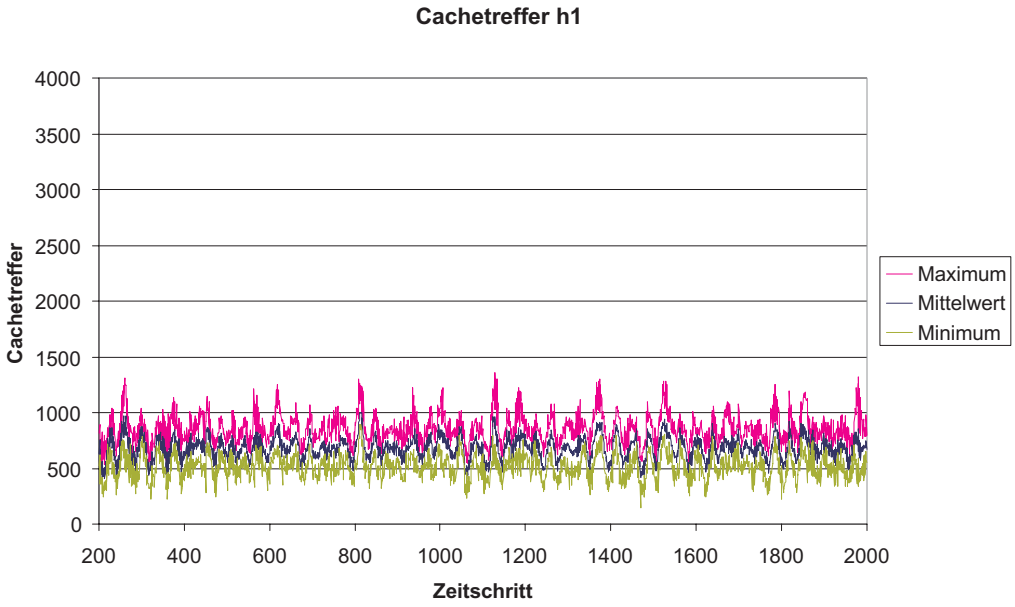

Abb. 10-9 Cachetreffer in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer Kombination der Breitensuche mit der Säulenpartitionierung (5)

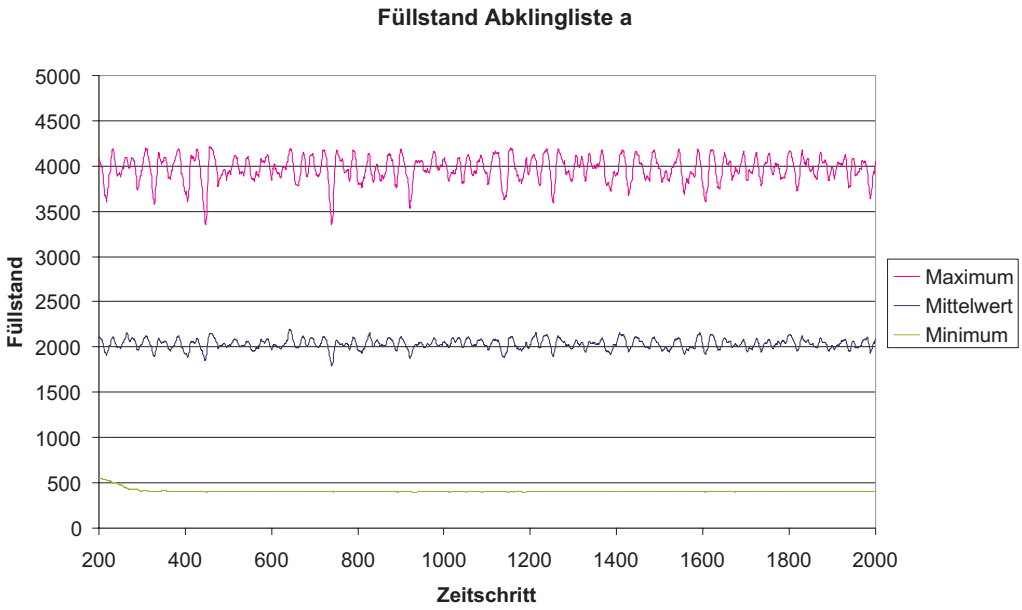

Abb. 10-10 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer lagenweisen Verteilung der Neuronen mittels der PARTY-Methoden (6)

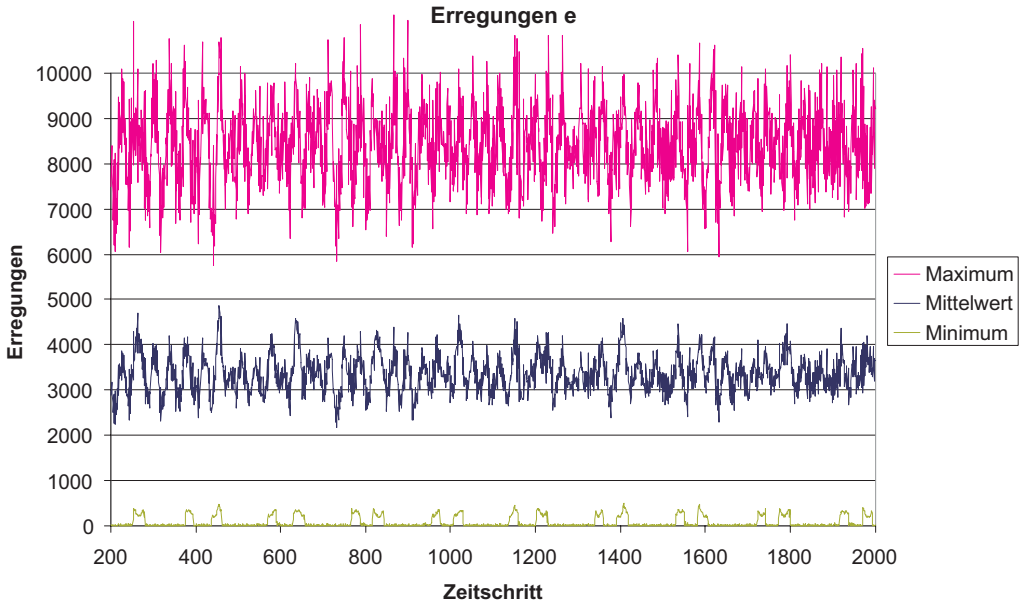

Abb. 10-11 Lastverteilung in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer lagenweisen Verteilung der Neuronen mittels der PARTY-Methoden (6)

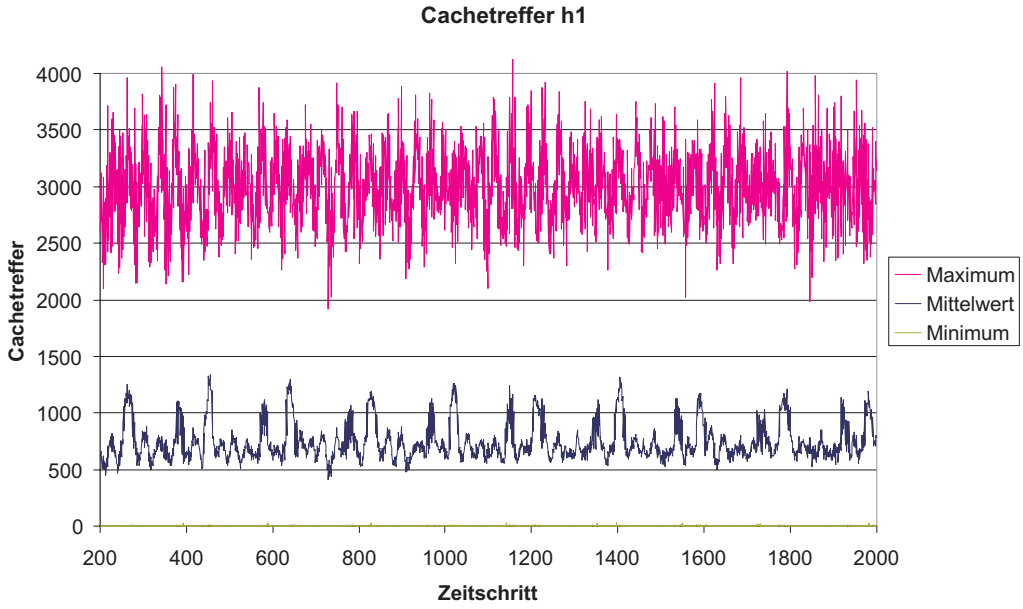

Abb. 10-12 Cachetreffer in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer lagenweisen Verteilung der Neuronen mittels der PARTY-Methoden (6)

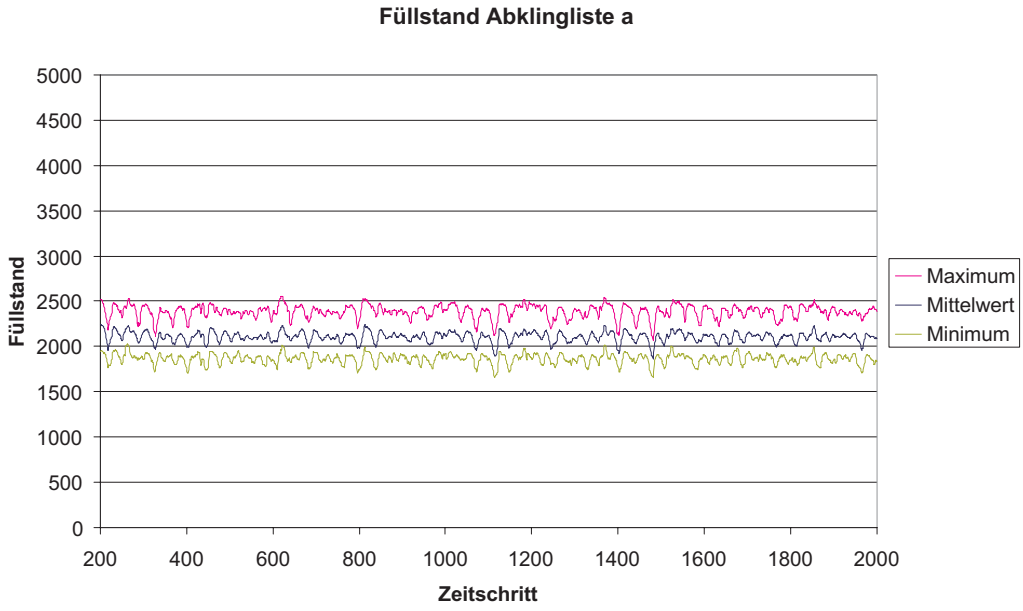

Abb. 10-13 Lastverteilung in der Abklingphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer Verteilung der Neuronen mittels der PARTY-Methoden und Säulen-Vorpartitionierung (7)

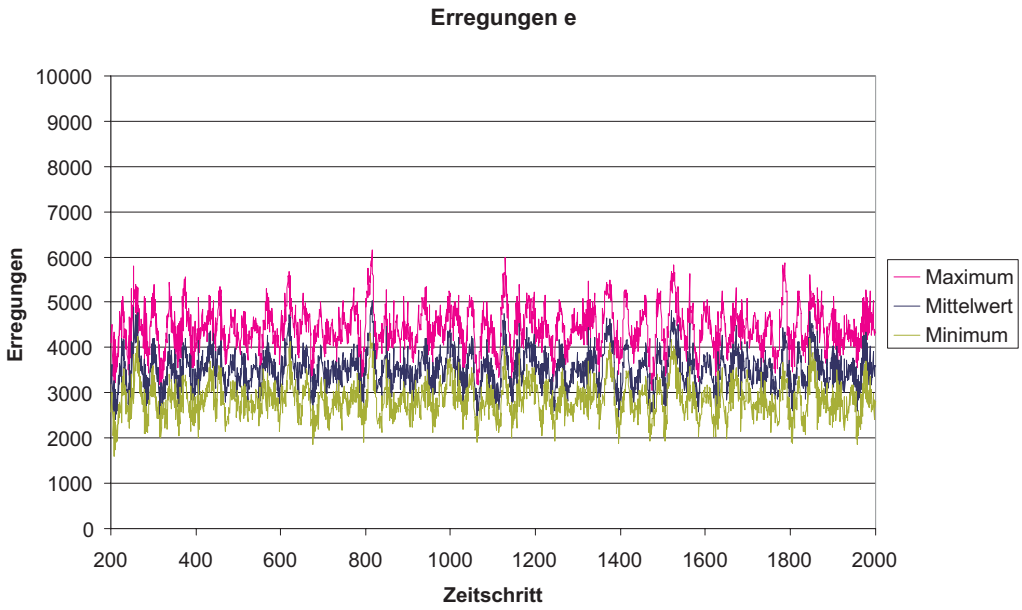

Abb. 10-14 Lastverteilung in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer Verteilung der Neuronen mittels der PARTY-Methoden und Säulen-Vorpartitionierung (7)

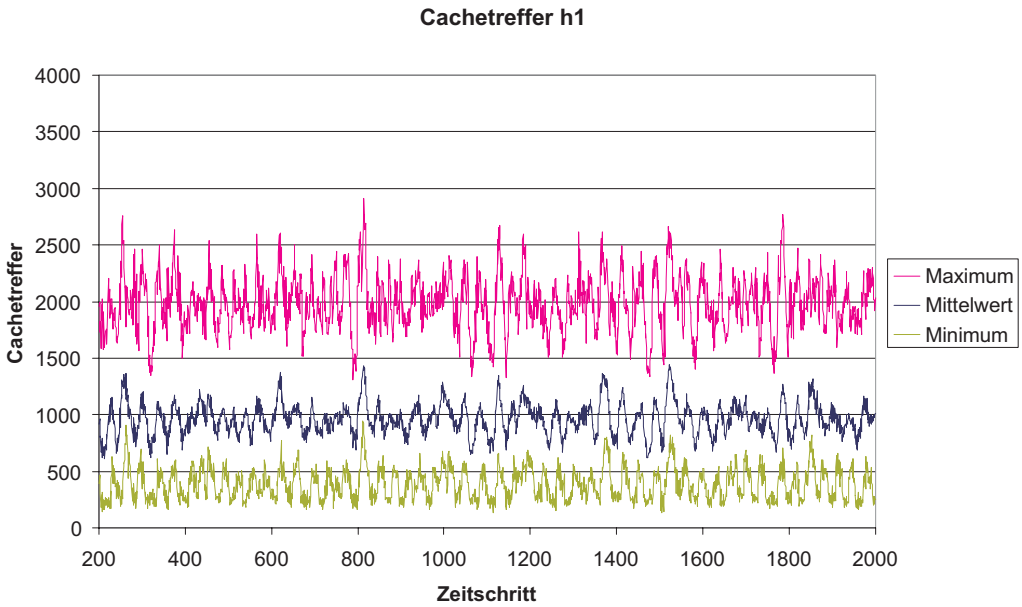

Abb. 10-15 Cachetreffer in der Erregungsphase für die Simulation des Weitzelnetzes auf 8 Maschinen bei einer Verteilung der Neuronen mittels der PARTY-Methoden und Säulen-Vorpartitionierung (7)