Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte

FZKA 6251

Optimierte Implementierung neuronaler Strukturen in Hardware

Thomas Fischer

Hauptabteilung Prozeßdatenverarbeitung und Elektronik

von der Fakultät für Elektrotechnik der Universität Karlsruhe
genehmigte Dissertation

Forschungszentrum Karlsruhe GmbH, Karlsruhe

1999

Optimierte Implementierung neuronaler Strukturen in Hardware

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der Fakultät für
Elektrotechnik
der Universität Fridericiana Karlsruhe

genehmigte

DISSERTATION

von

Dipl.-Ing. Thomas Fischer

aus Stuttgart

Tag der mündlichen Prüfung: 10. Dezember 1998

Hauptreferent: Prof. Dr.-Ing. K. D. Müller-Glaser

Korreferent: Prof. Dr. rer. nat. H. Gemmeke

Kurzfassung

In vielen Bereichen der Mustererkennung, wo eine exakte mathematische Systembeschreibung schwierig oder oftmals unmöglich ist, erweist sich der Einsatz künstlicher neuronaler Netze gegenüber klassischen Verfahren als sehr vorteilhaft. Werden neuronale Netze auf sequentiellen Standardprozessoren berechnet, bleibt ihre inhärente Parallelität ungenutzt, was sich insbesondere bei Anwendungen unter Echtzeitbedingungen als erheblicher Nachteil herausstellt. Wesentlich geeigneter sind Parallelprozessoren, die in Anlehnung an das biologische Vorbild künstlicher neuronaler Netze zahlreiche, oftmals sehr einfache Prozeßelemente in geeigneter Weise miteinander verknüpfen und damit wesentlich höhere Leistungswerte ermöglichen. Unverhältnismäßig hohe Kosten, mangelnde Flexibilität und eine äußerst komplizierte Handhabung solcher Parallelprozessoren verhinderten jedoch bisher einen breiten Einsatz.

Die vorliegende Arbeit befaßt sich mit der Konzeption und Realisierung einer Hardwarearchitektur zur Beschleunigung neuronaler Netze, die sich durch eine hohe Flexibilität, einfache Handhabung und geringe Herstellungskosten auszeichnet. Insbesondere der letztgenannte Punkt erfordert anstelle einer Fullcustom-Technologie den Einsatz einer Gate-Array-Lösung, die bei kleinen Stückzahlen erhebliche Kostenvorteile bringt.

Unter Berücksichtigung der technologischen Randbedingungen wurde eine neue Architektur entwickelt, die auf den Einsatz lokaler Speicher auf dem Chip verzichtet und gleichzeitig im Vergleich zu bekannten Architekturen weniger Busse zu externen Speichern erfordert. Kern dieser Architektur ist ein hybrides Kommunikationsschema, das sowohl Elemente systolischer Felder, als auch paralleler SIMD-Strukturen beinhaltet. Der reduzierte Kommunikationsaufwand und die Architektur der parallelen Prozeßelemente ermöglichen eine transparente Programmierung, die einer automatischen Code-Generierung sehr entgegenkommt.

Im Rahmen dieser Arbeit wurde die Implementierung dieser neuen Architektur auf der Basis eines Sea-Of-Gates Semicustom-Chips zur Beschleunigung von Operationen der Recall-Phase neuronaler Netze durchgeführt. Der hierbei entstandene anwendungsspezifische Prozessor namens SAND/1 (**S**imple **A**pplicable **N**eural **D**evice) erreicht mit vier parallelen Prozeßelementen bei 50 MHz Taktfrequenz eine maximale Leistung von 200 MCPS (Millionen Verbindungen pro Sekunde), was bis zu 600 MOPS (Millionen Operationen pro Sekunde) entspricht. Zur Beschleunigung PC-basierter Anwendungen wurde eine Einsteckkarte für den PCI-Bus entwickelt, die einen Parallelbetrieb von bis zu vier solcher SAND/1-Chips erlaubt und damit eine maximale Leistung von 2,4 GOPS (Giga Operationen pro Sekunde) erreicht.

Optimized Implementation of Neural Structures in Hardware

Abstract - In many applications in the field of pattern recognition it is often difficult or nearly impossible to describe the system behavior in an exact mathematical style. Therefore, neural networks seem to be well suited showing significant advantages compared to traditional algorithms. The inherent parallelism of neural networks can't be used if they are applied to standard sequential processors. This is a real disadvantage in the case of real-time applications. Specialized parallel processors with communication structures inspired by their biological model combine lots of simple processing elements to increase processing speed. Until now these processors had the common problem of being very expensive, inflexible and showed a difficult handling which results in a rare usage.

This work describes the design and implementation of a new architecture for speeding up neural calculations in hardware. This new architecture shows a simple handling and its implementation can be done at a very low cost level. Therefore a gate array technique is used, instead of expensive fullcustom technology which allows significant cost reduction, especially for small quantities.

Taking into account the limitations of a gate array design this new architecture was designed. It uses no local memory on the chip and requires less busses to external components, compared to traditional designs. This architecture is based on a new communication style which uses elements of both systolic arrays and one dimensional SIMD processing structures. The reduction of communication allows simple programming and the usage of automatic code generation tools.

In the framework of this dissertation, the new architecture has been implemented in hardware, based on a low cost sea of gates technology for speeding up neural calculations used within the recall phase of neural networks. This application specific processor is called SAND/1 (**s**imple **a**pplicable **n**eural **d**evice) and combines four parallel processing elements on one chip. At 50 MHz clock frequency it reaches a peak performance of 200 MCPS (million connections per second), which is equivalent to up to 600 MOPS (million operations per second). For speeding up PC based applications a PCI board was developed that allows parallel usage of up to four SAND chips. Maximum performance of this board is 2.4 GOPS (giga operations per second).

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als Doktorand am Forschungszentrum Karlsruhe in der Hauptabteilung Prozeßdatenverarbeitung und Elektronik (HPE).

Mein besonderer Dank gilt dem Leiter der HPE, Herrn Prof. Dr. H. Gemmeke, der mir die Möglichkeit zur Durchführung dieser Arbeit gab. Sein großes Interesse am Fortgang derselben hat wesentlich zu meiner Motivation und zum Gelingen dieser Dissertation beigetragen. In gleicher Weise möchte ich Herrn Prof. Dr. K. D. Müller-Glaser (Institut für Technik der Informationsverarbeitung, Universität Karlsruhe) für die Übernahme des Hauptreferats und die vielen wertvollen Hinweise und Ratschläge bei der Prüfungsvorbereitung danken.

Dem Leiter der Neuro-Fuzzy-Gruppe, Herrn Dr. W. Eppler, danke ich für die zahlreichen Diskussionen und Anregungen, die kompetente Unterstützung bei allen „Neuro-Problemen“, das äußerst gewissenhafte Korrekturlesen und das tolle Arbeitsklima in der Neuro-Fuzzy-Gruppe.

Ohne die hervorragende Zusammenarbeit mit unseren Kooperationspartnern IMS und datafactory hätte ein so umfangreiches Projekt wie „SAND/1“ nicht realisiert werden können. Dafür möchte ich mich beim Leiter des Instituts für Mikroelektronik der Universität Stuttgart (IMS), Herrn Prof. Dr. B. Höfflinger, und dem Geschäftsführer der datafactory GmbH, Herrn Thomas Becher, ganz herzlich bedanken.

Mein Dank gilt auch all denjenigen, die durch ihre fachliche und moralische Unterstützung zum Gelingen dieser Arbeit beigetragen und mir beim Überwinden vieler Hürden geholfen haben: Alexandre Menchikov, Matthias Balzer, Dr. Gerd Kock und Dr. Stefan Neußer. Nicht zu vergessen sind die ehemaligen Diplomanden Bernd Rothenberger und Alexander Schreiber, die durch ihre Arbeiten einen wesentlich Beitrag zum Gelingen dieser Dissertation geleistet haben.

Unseren Mammographie- und JAVA-Experten Dr. Rainer Stotzka, Tim Müller, Volker Hartmann, Hansjörg Neiber und Sascha Damnig danke ich für die erfolgreiche Zusammenarbeit bei der Integration von SAND in die Mammographie-Workstation.

Nicht zuletzt das tolle Arbeitsklima der HPE und insbesondere der Neuro-Fuzzy-Gruppe hat dazu beigetragen, daß mir die Arbeit sehr viel Spaß bereitet hat - dafür möchte ich mich bei allen Kolleginnen und Kollegen ganz herzlich bedanken.

Für die finanzielle Unterstützung in Form eines Doktorandenstipendiums danke ich dem Forschungszentrum Karlsruhe.

Karlsruhe, im Dezember 1998 Thomas Fischer

Inhaltsverzeichnis

Abkürzungen und Formelzeichen

Abkürzungen

ACK	**Ack**nowledge
ACT	**A**ir **C**herenkov **T**elescope
ALU	**A**rithmetic **L**ogic **U**nit
ASIC	**A**pplication **s**pecific **i**ntegrated **c**ircuit
CCD	**C**harged **C**oupled **D**evice
DG	**D**atenabhängigkeits**g**raph
DISTL2	**Dist**anz 2. Ordnung / **L2**-Norm
DMA	**D**irect **M**emory **A**ccess
DS	**D**aten**s**atz
DSC	**D**ata **S**tream **C**ontroller
DSP	**d**igitaler **S**ignal**p**rozessor
FA	**f**ull **a**dder
GCPS	**G**iga **C**onnections **p**er **S**econd
IMS	**I**nstitut für **M**ikroelektronik **S**tuttgart
KOH	**Koh**onen-Netzwerk
LUT	**L**ook-**U**p-**T**abelle
MAC	**M**ultiply **Ac**cumulate
MAGIC	**M**ajor **A**tmospheric **G**amma **I**maging **C**herenkov **T**elescopes
MCPS	**M**ega **C**onnections **p**er **S**econd
MIMD	**M**ultiple **I**nstruction Streams - **M**ultiple **D**ata Streams
MIPS	**M**illionen **I**nstruktionen **p**ro **S**ekunde
MISD	**M**ultiple **I**nstruction Streams - **S**ingle **D**ata Stream
MLP0...3	**M**ulti**l**ayer-**P**erzeptron mit **0 ... 3** verdeckten Schichten
MMAC/s	**M**illionen **MAC**-Operationen pro **S**ekunde
MMX	**M**ulti**m**edia E**x**tensions
MOPS	**M**illionen **O**perationen **p**ro **S**ekunde
NN	**N**euronales **N**etz
PCI	**P**eripheral **C**omponent **I**nterconnect
PE	**P**rozeß**e**lement
PN	**P**rocessor **N**ode
RBF	**R**adiale-**B**asis**f**unktionen-Netze
ROI	**R**egions **o**f **i**nterest
SAND	**S**imple **A**pplicable **N**eural **D**evice
SEQ	**Seq**uencer
SIMD	**S**ingle **I**nstruction Stream - **M**ultiple **D**ata Streams
SISD	**S**ingle **I**nstruction Stream - **S**ingle **D**ata Stream
SOM	**S**elf **O**rganizing **M**ap
SSE	**S**ummed **S**quare **E**rror
TDNN	**T**ime **D**elay **N**eural **N**etwork
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage
VHSIC	**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
VLIW	**V**ery **L**ong **I**nstruction **W**ord
VLSI	**V**ery **L**arge **S**cale **I**ntegration

Formelzeichen

$a,\ a_j$	Aktivierung eines Neurons
$\underline{a}$	Aktivierungsvektor einer gesamten Schicht
$A,\ A_{FA},$ $A_{parallel},\ A_{seriell}$	Fläche [mm^2]
$b,\ b_{ges},\ b_{parallel},$ $b_{ges,\,max},\ b_{max,eff}$	Beschleunigung
$\underline{B}_1,\ \underline{B}_2$	Bildmatrix
$b_1,\ b_2$	Einzelner Bildpunkt einer Bildmatrix
$\underline{\tilde{b}}_1,\ \underline{\tilde{b}}_2$	Bildvektor
$\tilde{b}_1,\ \tilde{b}_2$	Einzelner Bildpunkt eines Bildvektors
c_{ji}	Gewicht in der Ausgangsschicht bei RBF-Netzen
$d,\ d_{NN},\ d_{Feld}$	Datendurchsatz
distL1, distL2, distLSup	Abstand zweier Vektoren
$\mathbf{D}_{ActIn},\ \mathbf{D}_{Transfer}$	Definitionsbereich
err_j	Fehler eines Neurons beim Training
$\underline{err}^{[v]}$	Fehlervektor einer Schicht beim Training
$E,\ E^{\min},\ E_{Feld}$	Effizienz
$f,\ f_{max},$ $f_0,\ f_{parallel},\ f_{seriell}$	Frequenz [Hz]
$f(a),\ f_{sigmoid},$ $f_{logistisch},\ f_{Gauß}$	Aktivierungsfunktion
h(x), g(x), f(x)	Funktionen des verallgemeinerten Neurons
$\underline{H},\ \underline{H}^{high},\ \underline{H}^{low},$ $\underline{H}^{band}$	Filtermaske (Matrix)
$h_{j,i}$	Einzelner Koeffizient einer Filtermaske
$k,\ k_{PE},\ k_C$	Anzahl paralleler Funktionseinheiten
l	Anzahl der Schichten eines Netzes
m	Anzahl der Neuronen in der zu berechnenden Schicht
n	Anzahl der Neuronen in der Sendeschicht
$N,\ N_{DIF},\ N_{MUL},$ $N_{WGT},\ N_{RES},$ $N_{SUM.e},\ N_{weights},$ N_{act}	Wortbreite [Bit]
$\underline{o}^{[v]}$	Aktivitätsvektor einer Schicht
$\underline{O}^{[v]}$	Aktivitätsmatrix einer Schicht

P_{ges}	Leistungsaufnahme [W]
p_{MCPS}, p_{MCPPS}^{abs}, p_{MCPPS}^{16Bit}, $p_{MCPPSPE}^{16Bit}$	Rechenleistung
S, S_h	Anzahl der Schritte bei der Abbildung eines Netzes auf einen Chip
S'	Anzahl der Schritte bei der Abbildung eines Netzes auf mehrere Chips
t_l, t_l^{neu}, t_l^{Feld}, t_l^{BC}	Latenzzeit (normiert auf die Taktzykluszeit des Prozessors)
t_Σ, t_f, t_{ges}, t_{Host}, t_{Nutz}, t_{ctrl}, t_{idle}, t_{read}	Auf die Taktzykluszeit eines Prozessors normierte Zeitangabe
T, T_P, T_{FA}	Taktzykluszeit [s]
U	Anzahl sämtlicher Trainingsmuster
V_{dd}, V_I, V_O	Spannungen [V]
w_{ji}, $w_{ji}^{[\nu]}$	Gewicht einer synaptischen Verbindung
$\underline{w}_j$	Gewichtsvektor eines Neurons
$\underline{W}$, $\underline{W}_o$, $\underline{W}_h$, $\underline{W}^{[\nu]}$	Gewichtsmatrix einer gesamten Schicht
δ_{aj}, δ_{oj}	Quantisierungseinheit
Δw_{ji}	Gewichtsänderung einer einzelnen Verbindung beim Training
$\Delta_e \underline{W}^{[\nu]}$, $\Delta \underline{W}^{[\nu]}$	Gewichtsänderungen (Matrix) einer gesamten Schicht beim Training
$\Delta\Theta_j$	Schwellwertänderung beim Training
$\Delta\tau$, $\Delta\tau_{calc}$, $\Delta\tau_{Latenz}$, $\Delta\tau_{ges}$	Absolute Differenzzeit [s]
η, $\eta(\nu)$	Lernrate
Θ_j	Schwellwert eines einzelnen Neurons
$\underline{\Theta}^{[\nu]}$	Schwellwertvektor einer gesamten Schicht
$\tau_{kritisch}$	Absolute Zeitangaben [s]

Verwendete Indizierungen

ν, [ν]	Index einer Schicht
e	Index eines Musters
i	Index eines Sendeneurons
j	Index eines Empfangsneurons
p	Zeilenindex einer Bildmatrix
q	Spaltenindex einer Bildmatrix

1 Einleitung

1.1 Motivation und Ziel dieser Arbeit

Aus dem Experimentierstatus längst herausgewachsen, befinden sich digitale neuronale Hardwarekomponenten auf der Basis paralleler Architekturen in Konkurrenz zu Standardprozessoren und müssen sich nicht nur in bezug auf die Leistung an den immer schneller werdenden General-Purpose-Prozessoren messen. Zahlreiche, mittlerweile längst kommerziell erhältliche digitale Neuro-Prozessoren erreichen zwar beachtliche Beschleunigungen gegenüber Standardprozessoren, wie sie in PCs und Workstations verwendet werden, doch liegen die Preise der spezialisierten Hardware-Komponenten um ein Vielfaches über denen der General-Purpose-Prozessoren. Sowohl höhere Taktfrequenzen, ermöglicht durch Verbesserungen im technologischen Sektor, als auch Erweiterungen der Architektur (MMX) tragen zu einer ständigen Leistungssteigerung der Standardprozessoren bei. Auch digitale Signalprozessoren unterliegen einem ständigen Entwicklungsprozeß und erreichen durch Nutzung paralleler Architekturkomponenten Leistungen, die bis vor wenigen Jahren noch teuren Supercomputern vorbehalten waren. Durch vielfältige Einsatzmöglichkeiten dieser Massenprodukte sinken beinahe täglich die Preise, und es stellt sich die berechtigte Frage, ob überhaupt noch spezialisierte Hardware für die Simulation neuronaler Netze notwendig und wirtschaftlich vertretbar ist [Cabestany96, Ienne95a].

Basis für die hohe Leistung digitaler Neuro-Prozessoren ist neben der Parallelverarbeitung die Optimierung der einzelnen Hardwarekomponenten auf der Grundlage eines Fullcustom-Designs, das allgemein bei der Entwicklung von Prozessoren üblich ist. Während sich die hieraus resultierenden hohen Fixkosten bei General-Purpose-Prozessoren auf einige hunderttausend, oftmals sogar einige Millionen Exemplare verteilen, ist der Markt für neuronale Hardware verschwindend gering, so daß sich infolgedessen vergleichsweise hohe Einzelpreise digitaler Neuro-Prozessoren ergeben.

Neben diesem Preis-Leistungs-Dilemma zeigen digitale Neuro-Prozessoren in der Praxis eine weitere Schwäche. Ihre Parallelverarbeitung ermöglicht zwar ungleich höhere Verarbeitungsleistungen als rein sequentiell arbeitende Prozessoren, sie erfordert aber wesentlich mehr Entwicklungsaufwand. Die Handhabung dieser Spezial-Prozessoren ist so komplex, daß ein Anwender ein fundiertes Wissen bezüglich der Parallelisierung neuronaler Netze besitzen muß oder zumindest Erfahrungen in der Programmierung von Parallel-Prozessoren haben sollte. In Zeiten immer kürzer werdender „Time-To-Market“-Zyklen werden solche Kostenfallen eliminiert, was dazu führt, daß digitale Neuro-Prozessoren, ungeachtet ihrer Leistung, kaum zum Einsatz kommen.

Ungeachtet dieser Nachteile gibt es nach wie vor zahlreiche Anwendungen, die den Einsatz einer schnellen Hardware zur Beschleunigung neuronaler Operationen erfordern (Kapitel 1.3). Zunächst lassen sich diese Anwendungen in zwei Gruppen unterteilen: in eine, deren Ausgangspunkt ein PC oder eine Workstation ist und in eine zweite Gruppe, die unabhängig von einer solchen Plattform oftmals nur einen Microcontroller oder DSP zur Signalverarbeitung besitzt (Stand-Alone-Anwendung). Eine Hardwarekomponente zur Beschleunigung neuronaler Netze muß für beide Gruppen gleichermaßen geeignet sein.

Ein ganz entscheidender Punkt ist die Handhabung solch einer Hardware-Komponente. Nur wenn es gelingt, die Handhabung eines Parallelprozessors so zu vereinfachen, daß auch ein Anwender ohne fundierte Kenntnisse der parallelen Hardware schnell und somit kostengünstig eine Anwendung entwickeln kann, wird ein solcher Prozessor eine hohe Akzeptanz finden. Im Hinblick auf den Einsatz im Rahmen einer Stand-Alone-Anwendung fällt unter diesen Punkt auch eine einfache Hardware-Schnittstelle mit nur einer geringen Anzahl zusätzlicher peripherer Komponenten.

Dem ständigen Preisverfall der Standardprozessoren kann nur dann wirksam entgegengetreten werden, wenn ein Spezialprozessor zur Beschleunigung neuronaler Netze ein äußerst günstiges Preis-Leistungsverhältnis besitzt. Dabei muß sichergestellt sein, daß die Leistung dieser Hardware-Komponente auch die der nächsten Prozessorgenerationen noch deutlich übertrifft und dennoch der Preis (vor allem bei kleinen Stückzahlen) auf einem sehr niederen Niveau liegt.

Keine der bekannten Architekturen bzw. deren Implementierung sind in der Lage, alle diese Forderungen (einfache Handhabung, hohe Leistung, günstiger Preis) gleichermaßen zu erfüllen, was sich anhand eines Überblicks über den Stand der Technik , von Zitaten aus der Literatur (Kapitel 3) und eigenen Untersuchungen (Kapitel 6.2) belegen läßt. Motiviert durch zahlreiche Anwendungen im Forschungszentrum Karlsruhe und die oben erläuterten Mängel entstand diese Arbeit, deren Ziel eine optimierte Implementierung neuronaler Strukturen in Hardware ist, um so die genannten Forderungen zu erfüllen.

Kernstück der Arbeit ist eine neue Architektur zur Beschleunigung neuronaler Netze und Operationen der Bildverarbeitung, bei deren Entwurf die oben erläuterten Randbedingungen Beachtung fanden. Die Optimierung umfaßt dabei sowohl das Kommunikationsschema paralleler Prozeßelemente als auch die Architektur der Prozeßelemente selbst, um hierdurch eine maximale Verarbeitungsleistung und eine einfache Handhabung zu ermöglichen.

Eine kritische Diskussion dieser Architektur, basierend auf einer konkreten Implementierung und Realisierung, soll die Stärken und Schwächen dokumentieren und darlegen, wo diese Architektur im Vergleich zu preiswerten, leistungsfähigen Standardprozessoren und digitalen Signalprozessoren einzuordnen ist.

1.2 Künstliche neuronale Netze und ihre Eigenschaften

Künstliche neuronale Netze sind aus der Motivation heraus entstanden, biologische neuronale Netze als informationsverarbeitende Systeme zu modellieren. Die Anfänge dieses noch relativ jungen Wissenschaftszweiges gehen auf das Jahr 1943 zurück, als die beiden Amerikaner McCulloch und Pitts in einem Artikel [McCul43] erläuterten, wie einfache Klassen von Netzen auf der Basis ihres „McCulloch-Pitts“-Neurons prinzipiell jede arithmetische oder logische Funktion berechnen können. Bereits kurze Zeit später, im Jahre 1949, veröffentlichte Donald O. Hebb die mittlerweile klassische Hebb’sche Lernregel [Hebb49]. Unter Leitung von Frank Rosenblatt entstand am MIT der erste Neuro-Computer namens „MARK I Perceptron“, der auf einem neuen Neuronenmodell, dem Perzeptron, basierte [Rosenb58]. Auch an der Universität Karlsruhe wurde in dieser Zeit bereits an der Implementierung neuronaler Netzstrukturen in Hardware gearbeitet. In seiner Arbeit „Die Lernmatrix“ stellte Karl Steinbuch im Jahre 1961 eine einfache technische Realisierung eines Assoziativ-Speichers

vor [Steinb61]. Durch die Veröffentlichung des Buchs „Perceptrons“ von Marvin Minsky und Seymour Papert im Jahre 1969 erlitt die noch junge Neurowissenschaft eine spürbare Lähmung. Die beiden Autoren analysierten das einschichtige Perzeptron und bewiesen dessen Unfähigkeit zur Lösung einiger wichtiger Aufgaben der Musterklassifikation. Nur wenige aufsehenerregende Arbeiten wurden in den folgenden Jahren bekannt, darunter diejenige von Teuvo Kohonen, dessen selbstorganisierende Karten [Koh82] auch heute noch für viele Anwendungen, z.B. bei der automatischen Ziffernerkennung, von großer Bedeutung sind. John Hopfields Veröffentlichungen über rückgekoppelte Netze zur Lösung von Optimierungsaufgaben 1985 und vor allem der Artikel „Learning representations by back-propagating errors“ von Rumelhart und McClelland im Jahre 1986 [Rum86] brachten die Wende.

Mittlerweile werden unter dem Begriff „künstliche neuronale Netze“ eine Vielzahl von Neuronen- und Netzwerkarchitekturen zusammengefaßt, deren Gemeinsamkeit darin besteht, den Zusammenhang von Ein- und Ausgangsdaten aus Beispielen zu erlernen. Diese Eigenschaft erweist sich besonders dann als sehr vorteilhaft, wenn ein nichtlinearer Zusammenhang zwischen Ein- und Ausgangsdaten besteht und eine exakte mathematische Modellierung schwierig ist, sei es aus mangelnden Kenntnissen über das Systemverhalten oder aufgrund von dessen hoher Komplexität. Lernverfahren wie z.B. Backpropagation sind in der Lage, in einem iterativen Prozeß die Verbindungsgewichte der Neuronen so einzustellen, daß das Netz sich dem gewünschten Verhalten annähert, ohne von vornherein eine exakte mathematische Beschreibung des Systemverhaltens zu kennen. Durch Vergleich mit einer erwünschten Netzausgabe werden unter Minimierung des Gesamtfehlers die Gewichte angepaßt. Liegt der Fehler unterhalb einer vorgegebenen Schwelle, dann hat das Netz seine Gewichte adaptiert und damit ein implizites Modell des zu modellierenden Systems erstellt.

Eine weitere Eigenschaft neuronaler Netze ist ihre inhärente Parallelität, die durch die Vernetzung einfacher Verarbeitungseinheiten (Neuronen) entsteht. Hierin liegt ein großes Beschleunigungspotential, das aber nur dann genutzt werden kann, wenn eine entsprechende Hardwarearchitektur zur Verfügung steht. Im Zusammenhang mit der Lernfähigkeit ergeben sich vielfältige Anwendungsmöglichkeiten neuronaler Netze z.B. im Bereich adaptiver Systeme unter Echtzeitbedingungen.

1.3 Einsatz neuronaler Netze bei zeitkritischen Anwendungen

Auf vielen Anwendungsgebieten genügt die Rechenleistung heutiger General-Purpose-Prozessoren, um komplexe Aufgabenstellungen mit Hilfe neuronaler Netze zu lösen. Schnelle Floatingpoint-Prozessoren wie z.B. der Pentium II oder der PowerPC erreichen Leistungen bis zu 10 MCPS (Millionen Verbindungen pro Sekunde)[1], wobei Versuche[2] zeigten, daß ein Durchschnittswert von 2,5 MCPS realistisch erscheint. Für Stand-Alone-Anwendungen werden oftmals 16-Bit-Mikrocontroller eingesetzt, deren geringe Rechenleistung für einfache Klassifikationsaufgaben ausreicht. Einige Aufgabenstellungen (z.B. Triggerexperimente in der Astroteilchenphysik) erfordern aber Leistungen, die von heutigen General-Purpose-Prozessoren bei weitem noch nicht erbracht werden können

[1] Eine genaue Definition erfolgt in Kapitel 2.4.2

[2] Siehe hierzu Kapitel 6

[Gemmeke98]. Die folgenden Beispiele verdeutlichen die Notwendigkeit einer Hardwarebeschleunigung für neuronale Netze und ihnen verwandte Verfahren der Bildverarbeitung, um eine für die Praxis tragbare Verarbeitungszeit zu gewährleisten.

Computerunterstütze Diagnose (Mammographie)
Bei der automatischen Auswertung von Mammogrammen zur Früherkennung von Brustkrebs [Müller98] werden digitalisierte Röntgenbilder (ca. 4800 x 3600 Bildpunkte) pixelweise gefiltert, um damit sogenannte Mikrokalzifikationen sichtbar zu machen. Diese nur einige µm große Kalkablagerungen können auf einen Tumor in einem frühen Stadium hinweisen. Als Ergebnis der rechnergestützten Auswertung erhält der Radiologe Hinweise auf kritische Gebiete (ROI - **R**egions **o**f **I**nterest), um zusätzlich zu seiner eigenen Untersuchung die Diagnose sicherer zu machen. Für die Filterung von Bildern der oben angegebenen Größe werden ca. $1{,}2 \cdot 10^9$ Multiply-Add-Operationen benötigt. Hierfür würde ein Standardprozessor mehr als acht Minuten benötigen, was im klinischen Alltag untragbar wäre. Um die Verarbeitungszeit auf unter 10 Sekunden zu reduzieren, müßte eine Verarbeitungsleistung von 100 - 200 MCPS vorausgesetzt werden. Das zusehends unter Kostendruck geratende Gesundheitswesen erlaubt jedoch nicht den Einsatz einer teuren Spezialhardware, so daß hier ein akzeptabler Preis eine wichtige Rolle spielt.

Prognose von Ozonwerten
Bei der regionalen Prognose von Ozonwerten werden mittels neuronaler Netze meteorologische Daten ausgewertet und daraus eine maximale Ozonkonzentration für die folgenden 24 Stunden einer Region (z.B. eines Bundeslandes) vorhergesagt [Becher96a, Becher96b]. Gegenüber klassischen Verfahren (Transport-/ Chemiemodell) entfällt die Berechnung einiger tausend Differentialgleichungen, wodurch auf den Einsatz teurer Supercomputer, wie sie oft bei der Auswertung meteorologischer Daten gebräuchlich sind, verzichtet werden kann. Dennoch müssen insgesamt 80 neuronale Netze unterschiedlicher Topologie ausgewertet werden, deren Verarbeitungszeit einen erheblichen Anteil an der gesamten Rechenzeit ausmacht. Eine Beschleunigung der neuronalen Operationen wirkt sich daher unmittelbar auf die Verfügbarkeit der Prognose aus.

Auswertung von Luftaufnahmen
Nach Expertenschätzungen befinden sich auf dem Boden der Bundesrepublik Deutschland noch immer einige hundert Blindgänger aus dem zweiten Weltkrieg, die mit fortschreitender Zeit eine zunehmende Gefahr darstellen. Glücklicherweise existieren beinahe flächendekkend Luftaufnahmen der alliierten Streitkräfte, welche dokumentieren, wo sich mögliche Blindgänger befinden könnten. Allein für das Bundesland Nordrhein-Westfalen sind dies mehr als 370.000 Bilder, so daß eine manuelle Auswertung nicht in Betracht kommt. Zur Lösung dieses Problems wurde von einem Ingenieurbüro in Münster ein Verfahren entwikkelt [Cruse97], das basierend auf neuronalen Netzen eine automatische Auswertung erlaubt. Während die Auswertung eines Bildes auf einer modernen Workstation noch mehr als sechs Stunden benötigt, soll sie mit Hilfe digitaler Neuro-Hardware in weniger als zehn Minuten bewältigt werden. Um den ganzen Bestand an Bildern in überschaubarer Zeit auszuwerten, müßten mehrere PCs mit einer Beschleunigerkarte ausgestattet werden, so daß auch hier sowohl die Kosten als auch die Handhabung entscheidende Faktoren sind.

Hochenergiephysik
Bei Shower-Experimenten der Astroteilchenphysik zur Untersuchung von Teilchen hoher Energie (10 ... 300 GeV) muß innerhalb kurzer Zeit eine Vielzahl von Meßwerten gespeichert und anschließend ausgewertet werden. Beim MAGIC-Projekt [Lorenz97] könnte die Trigger-Rate des Cherenkov-Teleskops (ACT) wesentlich erhöht werden, wenn bereits während der Aufnahme der Meßwerte eine Vorentscheidung darüber getroffen würde, ob das Ereignis überhaupt Relevanz besitzt. Anhand von Simulationen konnte gezeigt werden, daß sich hierfür neuronale Netze eignen. Der Online-Einsatz erfordert aber Antwortzeiten innerhalb einiger 10 µs, die sich weder mit schnellen Standardprozessoren noch mit Signalprozessoren erreichen lassen, so daß hier der Einsatz schneller Neuro-Hardware notwendig ist [Gemmeke96]. Ähnliche Anforderungen bestehen im Auger-Experiment. Hierbei sollen für die Online-Mustererkennung innerhalb von 10 bis 100 µs sehr hochenergetische Teilchenpaare (10^{18} bis 10^{21} eV) in der Atmosphäre vom Hintergrund getrennt werden. Für dieses Experiment ist der Einsatz von 33 der im Rahmen dieser Arbeit entwickelten Neuro-Boards geplant.

1.4 Analoge und digitale Implementierungen

Für die Implementierung neuronaler Strukturen in Hardware eignen sich sowohl analoge als auch digitale Schaltungstechnik, die beide Vor- und Nachteile besitzen.

Analoge Schaltungstechnik hat den Vorteil, daß eine direkte Kopplung mit einem Sensor erfolgen kann, was in zahlreichen Anwendungen in bezug auf die Miniaturisierung erhebliche Vorteile bringt (künstliche Cochlea [Brucke98], künstliche Retina [Schlüß97]). Die Implementierung arithmetischer Funktionen auf der Basis analoger Schaltungstechnik erfordert weit weniger Chip-Fläche als eine äquivalente digitale Implementierung. Aktivitäten werden im allgemeinen als Ströme aufgefaßt, deren Multiplikation mit Gewichten nur wenige Transistoren erfordert [Jabri96]. Eine nichtlineare Transferfunktion läßt sich mit relativ geringem Aufwand auf der Basis eines n-MOS-Transistors realisieren, dessen Kennlinie ein deutlich nichtlineares Verhalten mit Sättigungscharakteristik aufweist, sofern der Transistor im Subthreshold-Bereich betrieben wird [Boahen89]. Schließlich ist die Verarbeitungsgeschwindigkeit analoger Implementierungen bei vergleichsweise sehr geringem Energieverbrauch um Größenordnungen höher als bei digitaler Schaltungstechnik. Diesen Vorteilen stehen einige grundlegende Nachteile gegenüber. Aufgrund des Rauschens in elektronischen Schaltungen und einer begrenzten Dynamik liegt die Auflösung analoger neuronaler Hardwarekomponenten unter acht Bit - insbesondere beim Lernen ergeben sich hier zum Teil erhebliche Nachteile. Schwankungen in der Versorgungsspannung und elektromagnetische Einstreuungen auf einer Platine können zu erheblichen Störungen und Fehlverhalten führen. Dies muß insbesondere bei einem industriellen Einsatz unter sicherheitsrelevanten Aspekten berücksichtigt werden. Auch die Speicherung der Gewichte gestaltet sich aufwendiger als bei digitalen Implementierungen. Häufig werden die Gewichte trotz analoger Verarbeitung digital außerhalb des Analogchips gespeichert und mittels D/A-Wandler in ein analoges Signal umgewandelt, so daß zusätzlicher Schaltungsaufwand notwendig ist.

Wenngleich die digitale Schaltungstechnik wesentlich aufwendiger und langsamer erscheint, bietet sie eine ungleich höhere Genauigkeit (Anzahl der Bit eines Operanden) und Flexibilität in Bezug auf die zu implementierenden Funktionen. Auch bezüglich der zur Verfügung stehenden Entwicklungswerkzeuge bietet die digitale Schaltungstechnik Vorzüge.

Für Anwendungen, bei denen in erster Linie eine hohe Integrationsdichte, geringer Energieverbrauch und leichte Kopplung mit analogen Sensoren wie z.B. CCD-Arrays gefordert werden, die erzielbare Genauigkeit aber eine untergeordnete Rolle spielt, stellen analoge Implementierungen die bessere Wahl dar. Für einen flexiblen Einsatzbereich, robustes Verhalten gegenüber Störungen und hohe Genauigkeit sind digitale Implementierungen den analogen vorzuziehen. Für die in der Einleitung skizzierten Anwendungsfelder ist daher die digitale Schaltungstechnik geeigneter, so daß sich die folgenden Betrachtungen und Aussagen hierauf beschränken.

1.5 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in insgesamt acht Kapitel. Dieser Einleitung folgt die Zusammenstellung einiger Grundlagen bezüglich der Implementierung neuronaler Netze in digitaler Hardware (Kapitel 2). Dieser Teil umfaßt die Klassifikation und einige Grundlagen der Parallelisierung neuronaler Netze sowie Grundlagen über parallele Rechnerstrukturen für die Beschleunigung neuronaler Netze. Das dritte Kapitel gibt einen Überblick über den Stand der Technik bei digitalen Chips zur Beschleunigung neuronaler Netze, wobei jene Implementierungen detaillierter betrachtet werden, die vor allem in bezug auf Rechenleistung bzw. Integrationsdichte die Spitze derzeit verfügbarer Neuro-Chips darstellen. Das vierte Kapitel greift die bereits im dritten Kapitel geschilderten Probleme der verschiedenen Implementierungen auf und zeigt Möglichkeiten auf, wie eine optimierte Architektur zur Beschleunigung der gebräuchlichsten[3] neuronalen Netze aussehen kann. Im fünften Kapitel wird eine konkrete Implementierung und Realisierung dieser Architektur vorgestellt, welche neben einem Neuro-Chip eine PCI-Einsteckkarte und ein Softwarepaket zur Ansteuerung dieses Boards umfaßt. Der Neuro-Chip erhielt den Namen SAND (**S**imple **A**pplicable **N**eural **D**evice), was zum Ausdruck bringen soll, daß sich dieser Prozessor vor allem durch seine einfache Handhabung auszeichnet. Ausführliche Benchmarks im sechsten Kapitel dienen der Bewertung dieser Implementierung und zeigen, wo sie im Vergleich zu leistungsfähigen Standardprozessoren einzuordnen ist. Im siebten Kapitel wird eine Anwendung vorgestellt, die von der kosteneffektiven Beschleunigung durch den im Rahmen dieser Arbeit entwickelten Neuro-Chip profitiert. Mit einer Zusammenfassung und einem Ausblick auf zukünftige Entwicklungen endet die Arbeit.

[3] Siehe hierzu Kapitel 2.2

2 Grundlagen der Implementierung neuronaler Strukturen

2.1 Einleitung

Im einleitenden Kapitel dieser Arbeit wurde bereits die inhärente Parallelität neuronaler Netze als ein herausragendes Merkmal erwähnt. Bei genauerer Betrachtung zeigt sich, daß verschiedene Parallelitätsstufen existieren, deren genaue Analyse Basis für eine optimierte Implementierung in digitaler paralleler Hardware ist.

2.2 Klassifikation neuronaler Netze

Jährlich werden auf den Konferenzen und Tagungen mit dem Themenschwerpunkt „neuronale Netze“ zahlreiche neue Neuronen- und Netzwerkmodelle sowie neue oder von bereits bekannten Lernverfahren hergeleitete Trainingsalgorithmen vorgestellt. Wollte man möglichst viele dieser Netze und Trainingsverfahren mit einer einzigen Architektur beschleunigen, so käme hierfür nur ein Prozessor mit General-Purpose-Charakter in Frage. Vor allem der CNAPS-10xx-Parallelprozessor von Adaptive Solutions[4] scheint am ehesten diesem Anspruch gerecht zu werden. Auf der anderen Seite legen zahlreiche Veröffentlichungen, die sich mit Anwendungen neuronaler Netze beschäftigen, die Vermutung nahe, daß sich ein Großteil der Applikationen auf nur wenige Neuronen- und Netzwerkmodelle mit den entsprechenden Lernverfahren stützt. Eine im Jahre 1995 vom Institut für Mikroelektronik Stuttgart (IMS) durchgeführte Analyse von mehr als 150 Veröffentlichungen über Anwendungen mit neuronalen Netzen ergab folgendes Ergebnis:

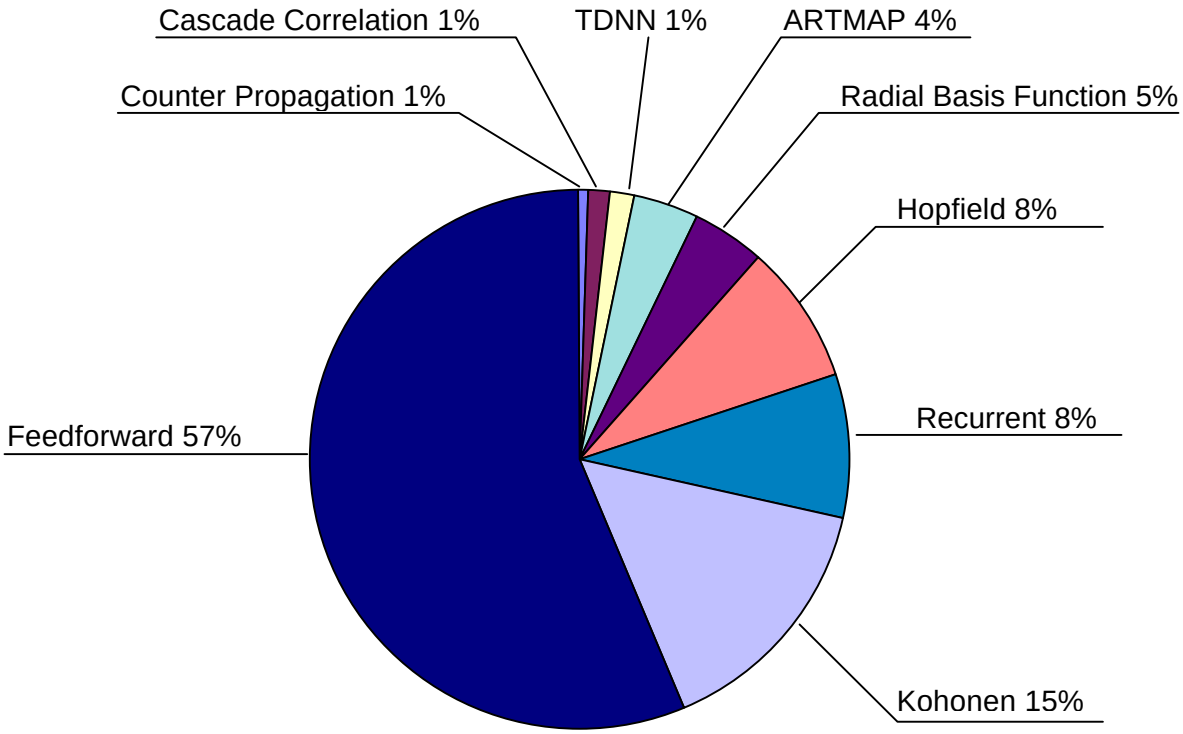


Abbildung 2.1: Marktanteil verschiedener Netzwerk-Typen

[4] Eine Ausführliche Betrachtung dieses Prozessors erfolgt in Kapitel 3

Bei den vorwärts gerichteten Netzen (Feedforward) handelt es sich im wesentlichen um Multilayer-Perzeptronen mit logistischer Aktivierungsfunktion, die häufig mit dem bereits erwähnten Backpropagation-Lernverfahren trainiert werden. Bei der zweitgrößten Klasse, den Kohonen-Netzen, handelt es sich um sogenannte selbstorganisierende Karten, die mit einem unüberwachten Lernverfahren trainiert werden. Die beiden in der Statistik folgenden Klassen (Recurrent und Hopfield) zeichnen sich durch eine Signalrückkopplung aus, die sie von allen anderen erwähnten Netzwerken unterscheidet. Neuronale Netze mit radialer Basisfunktion (RBF) besitzen wie Multilayer-Perzeptronen einen vorwärtsgerichteten Informationsfluß, wobei die Neuronen in der verdeckten Schicht eine Gauß'sche Funktion als Aktivierungsfunktion verwenden.

Im folgenden werden die Grundlagen der Netze kurz rekapituliert, die für die weiteren Betrachtungen in dieser Arbeit erforderlich sind. Didaktisch gut aufgebaute Einführungen in die Thematik neuronaler Netze findet man beispielsweise bei Zell [Zell94], Rojas [Rojas93] und Brause [Brause95].

Ein- und mehrschichtige Netze mit vorwärts gerichtetem Informationsfluß

Kennzeichnend für Netze dieser Klasse ist die Zusammenfassung von Neuronen in Schichten mit der Einschränkung, daß nur Verbindungen zwischen unmittelbar benachbarten Schichten in einer für das gesamte Netz einheitlichen Richtung existieren. Ferner wird vorausgesetzt, daß es keine Verbindungen zwischen Neuronen derselben Schicht gibt.

Multilayer-Perzeptron (MLP)

Auf der oben beschriebenen Topologie bauen die von Rumelhart et al. [Rum86] in Zusammenhang mit dem Backpropagation-Lernverfahren betrachteten Netze auf, deren Neuronen (Abbildung 2.2) eine Aktivierungsfunktion mit sigmoider Charakteristik (Abbildung 2.3) verwenden.

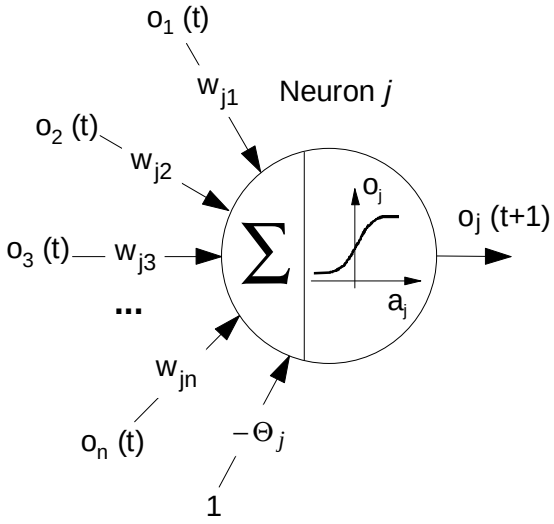


Abbildung 2.2: Neuron eines Multilayer-Perzeptrons

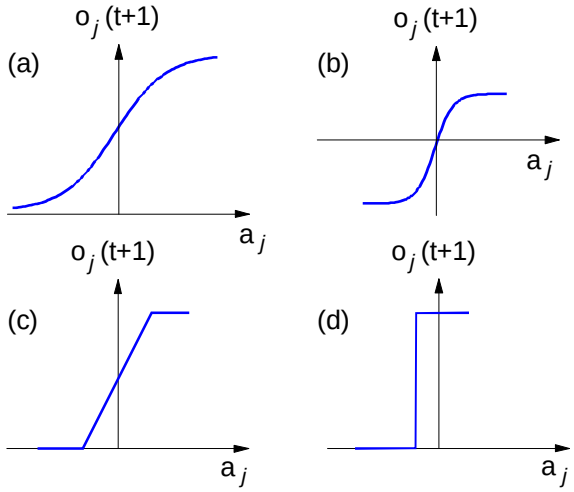


Abbildung 2.3: Sigmoide Aktivierungsfunktionen: (a) logistische Funktion, (b) Tanges-Hyberbolicus, (c) stückweise lineare Funktion und (d) Stufenfunktion.

Die mathematische Beschreibung der in Abbildung 2.2 angedeuteten Funktionsweise eines Neurons geht aus den Gleichungen (2.1), (2.2) und (2.3) hervor.

$$a_j = \sum_{i=1}^{n} w_{ji} \cdot o_i(t) - \Theta_j \qquad (2.1)$$

$$o_j(t+1) = f_{sigmoid}\left(a_j\right) \qquad (2.2)$$

$$f_{sigmoid}^{logistisch} = \frac{1}{1+e^{-b \cdot a_j}} \qquad (2.3)$$

Ein häufig verwendetes Lernverfahren ist das bereits erwähnte Backpropagation. Sogenannte Trainingsdaten werden durch das Netz propagiert und am Ausgang mit zu erwartenden Sollwerten (t_j) verglichen. Die Differenz aus erwarteten und durch das Netz berechneten Ausgangsdaten wird als Fehler (err_j) aufgefaßt (2.4), welcher anschließend wieder rückwärts durch das Netz zum Eingang hinpropagiert wird. Diese Rückpropagierung dient der Anpassung von Gewichten (w_{ji}) und Schwellwerten (Θ_j). In jeder Schicht des Netzes kann mit dem Ziel, den Fehler zu minimieren, aus dem zurückpropagierten Fehler unter Berücksichtigung einer sogenannten Lernrate (η) die Veränderung der Gewichte berechnet werden (2.5), (2.6). Das Netz hat eine Aufgabe gelernt, wenn das globale Fehlerminimum erreicht ist. Insbesondere bei der Implementierung in Hardware ergeben sich in Zusammenhang mit dem Lernen einige Schwierigkeiten, welche in einem der folgenden Kapitel noch genauer betrachtet werden.

$$err_j = \begin{cases} f'_{sigmoid}\left(a_j\right) \cdot \left(t_j - o_j\right) & \textit{für die Ausgangsschicht} \\ f'_{sigmoid}\left(a_j\right) \cdot \left(\sum err_k \cdot w_{kj}\right) & \textit{für verdeckte Schichten} \end{cases} \qquad (2.4)$$

$$\Delta w_{ji} = \eta \cdot err_j \cdot o_i \qquad (2.5)$$

$$\Delta \Theta_j = \eta \cdot err_j \qquad (2.6)$$

Selbstorganisierende Karten (SOM, Kohonen Feature Maps)

Die als Kohonen-Karten bekanntgewordenen Netze gehen auf Teuvo Kohonen zurück, der hierüber im Jahre 1982 erstmals eine Veröffentlichung verfaßte [Koh82]. Kennzeichnend für diese Klasse von Netzen ist eine einschichtige Topologie, die mit einem unüberwachten Lernverfahren trainiert wird. Ziel dieses Trainings ist es, in Lerndaten selbständig Cluster zu finden und unbekannte Eingabemuster einem dieser Cluster zuzuordnen. Dabei wird der Abstand zwischen einem Eingangsvektor

$$\underline{o}(t) = \left(o_1(t), \ldots, o_i(t), \ldots, o_n(t)\right)^T \qquad (2.7)$$

und den als Vektor zusammengefaßten Gewichten eines jeden Neurons (Index *j*)

$$\underline{w}_j = \left(w_{j1}, \ldots, w_{jn}\right) \qquad (2.8)$$

in der sogenannten Kohonenschicht berechnet. Dasjenige Neuron, welches den geringsten Abstand zum Eingabevektor besitzt, ist der Gewinner. Im Gegensatz zu Multilayer-Perzeptronen existieren bei Kohonen-Netzen lokale Beziehungen zwischen benachbarten

Neuronen. Dies bedeutet beim Training, daß unmittelbar benachbarte Neuronen eines Gewinners ebenfalls beeinflußt werden. Wie stark diese Beeinflussung ausfällt, hängt vom Abstand und der gewählten Nachbarschaftsfunktion ab. Häufig wird die normalisierte Gaußfunktion verwendet. In der Recallphase werden dem trainierten Netz dann (unbekannte) Eingabemuster präsentiert. Eine Einordnung in die im Rahmen des Trainings ermittelten Clustermittelpunkte (=Prototypvektoren) erfolgt derart, daß dasjenige Neuron, dessen Gewichtsvektor dem aktuellen Muster am ähnlichsten ist, aktiviert wird. Diese Ähnlichkeit wird anhand des Abstandes der beiden betrachteten Vektoren ermittelt, d.h. mathematisch gesehen wird der kleinste Abstand ermittelt (2.10). Wie schon beim Training, wird hierfür häufig die euklidische Distanz (L2-Norm) verwendet (2.9):

$$distL2_j = \left\| \underline{o}(t) - \underline{w}_j \right\|_{L2} = \sqrt{\sum_{i=1}^{n} \left(o_i(t) - w_{ji} \right)^2} \qquad (2.9)$$

$$n_{winner} = \arg\min_j \left(\left\| \underline{o}(t) - \underline{w}_j \right\|_{L2} \right) \ \forall \ j \in [1,m] \qquad (2.10)$$

Kohonen-Netze werden oft zum Auffinden von Ähnlichkeiten in hochdimensionalen Eingaberäumen verwendet. Typisches Anwendungsgebiet ist z.B. die Schrifterkennung.

Netze mit radialen Basisfunktionen (RBF)

Radiale-Basisfunktionen-Netze, im folgenden als RBF-Netze bezeichnet, besitzen grundsätzlich nur eine verdeckte Schicht (Abbildung 2.4). Die Neuronen dieser Schicht verwenden radialsymmetrische Aktivierungsfunktionen und stellen mathematisch gesehen ein Funktionensystem zur Approximation mehrdimensionaler Funktionen dar. Häufig wird, wie bei den Kohonen-Netzen, die Gaußfunktion (2.12) als Aktivierungsfunktion verwendet. RBF-Netze besitzen hierdurch gegenüber anderen Netzwerkmodellen wie z.B. MLPs einen wesentlichen Vorteil: aufgrund der radialsymmetrischen Basisfunktionen erzeugen RBF-Netze nur in der Nähe von Stützstellen (Zentren der Basisfunktionen) hohe Aktivierungen. Somit kann vermieden werden, daß Muster außerhalb des Trainingsbereiches unvorhersehbare Ausgaben liefern.

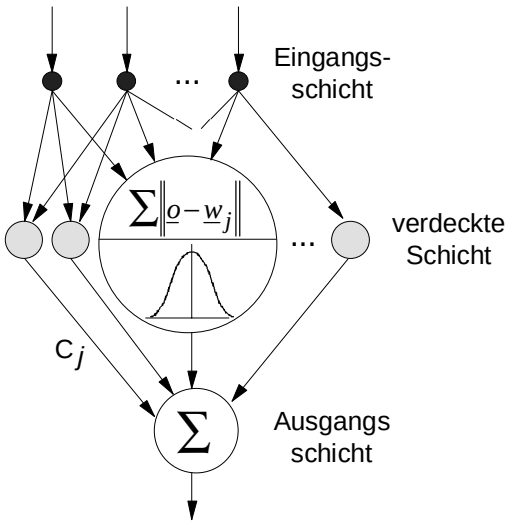


Abbildung 2.4: Beispiel eines RBF-Netzes

Die Aktivierung eines Neurons der verdeckten Schicht ist also umso höher, je näher das Eingangsmuster an einer Stützstelle liegt. Mathematisch gesehen entspricht diesem Maß der Abstand zwischen einer Stützstelle und einem zu bewertenden Punkt im *n*-dimensionalen Raum. Wie bei den Kohonen-Netzen wird dieser Abstand auf der Basis der L2-Norm (2.9) bestimmt. In der Ausgangsschicht werden dann sämtliche Aktivierungen der verdeckten Schicht bewertet, was im wesentlichen einer gewichteten Summation entspricht. Damit kann, mathematisch gesehen, ein RBF-Netz auf die zuvor betrachteten Strukturen MLP (gewichtete Summe) und Kohonen (Abstandsberechnung) zurückgeführt werden. Eine

Architektur, die für die Beschleunigung dieser beiden Netzwerktypen entworfen wurde, kann somit ohne zusätzlichen Aufwand auch für die schnelle Ausführung von RBF-Netzen in der Recall-Phase genutzt werden.

Zusammengefaßt kann die Ausgabe eines Neurons l in der Ausgangsschicht eines RBF-Netzes nach Gleichung (2.11) beschrieben werden.

$$o_l(t+1) = \sum_{j=1}^{m} c_j \cdot f_{Gauß}\left(\left\| \underline{o}(t) - \underline{w}_j \right\|_{L2} \right) \qquad (2.11)$$

$$f_{Gauß}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \qquad (2.12)$$

Rückgekoppelte Netze

Rückgekoppelte Netze unterscheiden sich von den bisher in dieser Arbeit betrachteten Netzen dadurch, daß die Ausgänge wieder auf die Eingänge zurückgekoppelt werden, was dazu führt, daß diese Klasse von Netzen nicht mehr zyklenfrei ist. Die nach J. Hopfield benannten Hopfield-Netze gehören ebenfalls zu dieser Klasse, wobei sie die Besonderheit aufweisen, daß es nur Verbindungen zu anderen Neuronen des Netzes, jedoch keine Verbindung zwischen Ein- und Ausgang desselben Neurons gibt. In der Praxis spielt die Stabilität von rückgekoppelten Netzen, die häufig für Optimierungsaufgaben eingesetzt werden, eine erhebliche Rolle. Um ähnlich wie bei Regelkreisen Oszillationen zu vermeiden, ist die Beachtung von Stabilitätskriterien äußerst wichtig.

Im Gegensatz zu Feedforward-Netzen, bei denen die Dauer einer Propagierung exakt bestimmbar ist, kann bei rückgekoppelten Netzen hierüber keine allgemeingültige Aussage getroffen werden. Im Vergleich zu den zuvor betrachteten Netzwerktypen (MLP, Kohonen und RBF) erfordert eine rückgekoppelte Netzwerktopologie ein wesentlich aufwendigeres Konzept für die Steuerung. Berücksichtigt man den relativ geringen Marktanteil dieser Netzwerktypen (Abbildung 2.1), dann scheint sich der höhere Implementierungsaufwand vor allem unter dem Gesichtspunkt der gesteckten Ziele (einfache Handhabung, optimierte Implementierung) nicht zu lohnen, so daß sich die folgenden Betrachtungen auf rückkopplungsfreie Netze beziehen werden.

2.3 Parallelisierung neuronaler Netze

Nordström und Svensson schlagen auf der Basis von Feedforward-Netzen mit Backpropagation-Training in [Nords92] eine Kategorisierung verschiedener Parallelitätsebenen vor. Diese Betrachtungsweise läßt sich leicht auch auf andere Neuronenmodelle anwenden. Hierzu wird in Anlehnung an [Rojas93] ein verallgemeinertes Neuron eingeführt, dessen Aufbau sich mit Hilfe dreier Funktion h, g und f beschreiben läßt:

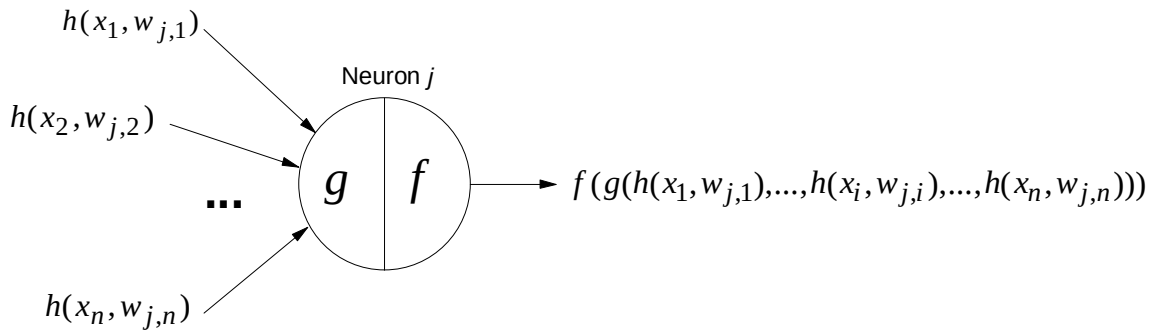


Abbildung 2.5: Verallgemeinertes Neuron zur Untersuchung verschiedener Parallelisierungsmöglichkeiten

Erläuterung zu den drei Funktionen h, g und f:

h Beschreibt die Verknüpfung von Eingangsaktivität und Kantenwert. Bei einem Multilayer-Perzeptron stellen die $w_{j,i}$ -Werte die Gewichte, bei einem RBF-Netz die Elemente des Prototypvektors dar.

g Beschreibt die Verknüpfung der durch die Funktion h abgebildeten Eingangswerte. Im Falle eines Multilayer-Perzeptrons entspricht g der Summation der Produkte aus Kantenwert und Eingangsaktivität.

f Aktivierungsfunktion, z.B. logistische Funktion im Falle eines Multilayer-Perzeptrons, bzw. Gauß'sche Funktion im Falle von RBF-Netzen.

Entsprechend der Klassifikation nach Kapitel 2.2 werden nur Netze mit vorwärtsgerichtetem Informationsfluß betrachtet. Die Anordnung der Neuronen erfolgt in H Schichten (H-1 verdeckte Schichten + eine Ausgangsschicht) wobei zur Vereinfachung angenommen werden kann, daß sich jede Schicht aus n Neuronen zusammensetzt. Unter Berücksichtigung dieser Randbedingungen lassen sich folgende Formen der Parallelisierung identifizieren:

Verbindungsparallelität (Stufe 0)

Bei dieser höchsten Form der Parallelität werden alle h-, g- und f- Funktionen parallel berechnet, wobei der Aufwand nahezu proportional zur Anzahl aller Verbindungen im Netz ist. Diese Form der Parallelität ermöglicht kürzeste Berechnungszeiten bei maximalem Hardwareaufwand. Um einen quantitativen Vergleich mit den anderen Parallelisierungsarten zu ermöglichen, sollen der zeitliche Aufwand und der Aufwand für die Hardware bestimmt werden.

Berechnungszeit	$O(1)$
Hardware-Aufwand	$O(n^2 \cdot H)$
Eigenschaften	+ schnellste Variante
	- hoher Kommunikationsaufwand (proportional zur Anzahl der Verbindungen im Netz), nimmt quadratisch mit der Anzahl der Neuronen zu
	- Hardware-Aufwand proportional zur Anzahl der Verbindungen des Netzes
	- schlechte Skalierbarkeit

Neuronenparallelität (Stufe 1)
Wird ein Neuron als kleinste Verarbeitungseinheit aufgefaßt und auf dieser Ebene eine Parallelisierung vorgenommen, dann liegt Neuronenparallelität vor. In diesem Fall werden alle h-, g- und f- Funktionen innerhalb eines Neurons sequentiell, die Neuronen des gesamten Netzes aber parallel berechnet. Eine Prozeßeinheit eines Prozessorverbundes entspricht hierbei einem Neuron.

Berechnungszeit	$O(n)$
Hardware-Aufwand	$O(n \cdot H)$
Eigenschaften	+ Aufwand proportional zur Anzahl der Neuronen
	+ bessere Skalierbarkeit als bei Stufe 0

Schichtparallelität (Stufe 2a)
Anstatt alle Neuronen eines gesamten Netzes parallel zu berechnen, können die Neuronen einer Schicht sequentiell und die Schichten selber parallel, z.B. in einer Pipeline, berechnet werden. In diesem Fall ist der Hardware-Aufwand proportional zur Anzahl der Schichten des Netzes.

Berechnungszeit	$O(n^2)$
Hardware-Aufwand	$O(H)$
Eigenschaften	+ geringer Hardwareaufwand
	- inhärente Parallelität neuronaler Netze wird kaum genutzt

Neben dieser horizontalen kann auch eine vertikale Aufteilung vorgenommen werden [Koll94], d.h. das Netz wird spaltenweise auf mehrere Rechenwerke abgebildet (Abbildung 2.6), so daß sich folgende Verhältnisse ergeben:

Berechnungszeit	$O(n \cdot H)$
Hardware-Aufwand	$O(n)$
Eigenschaften	+ für bestimmte Netzkonfigurationen vorteilhaft ($H << n$)
	Motivation: Kommunikationsstruktur bei Massiv-Parallel-Rechnern

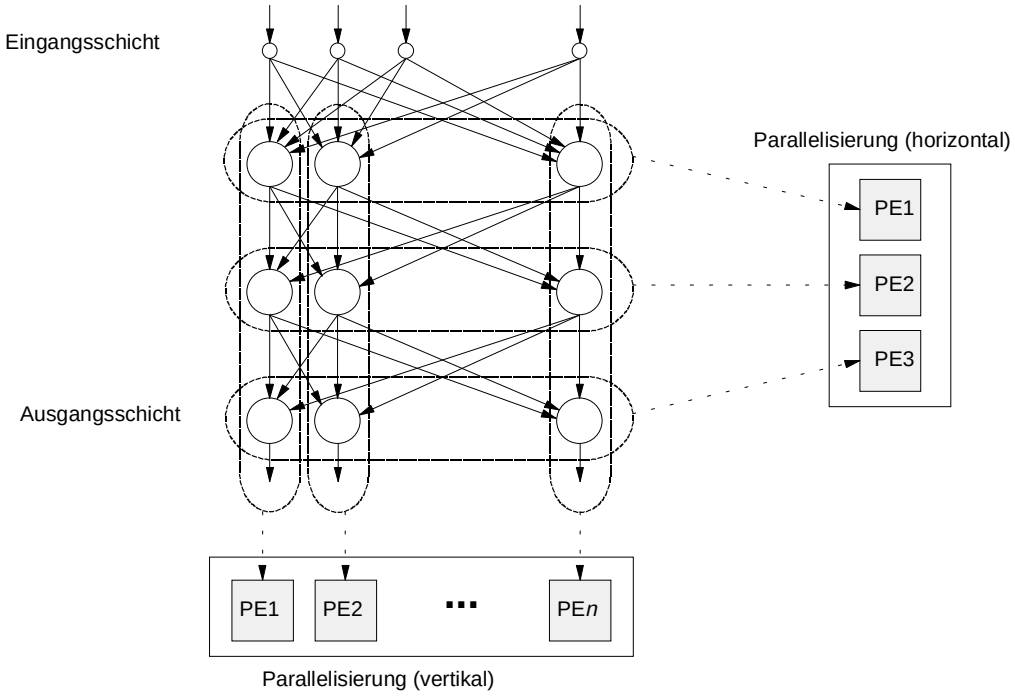


Abbildung 2.6: horizontale und vertikale Parallelisierung auf der Ebene einzelner Schichten (2a)

Neuronen-Schichtparallelität (Stufe 2b)

Anstatt die Schichten eines Netzes parallel und die darin zusammengefaßten Neuronen sequentiell zu berechnen, ist es auch möglich, nur eine Schicht eines Netzes in Hardware zu implementieren. D.h. die Schichten eines Netzes werden sequentiell und die darin zusammengefaßten Neuronen parallel berechnet.

Berechnungszeit	$O(n \cdot H)$
Hardware-Aufwand	$O(n)$
Eigenschaften	+ geringer Hardwareaufwand
	+ inhärente Parallelität neuronaler Netze wird besser ausgenutzt als bei Stufe 2a

Diese Stufe der Parallelisierung wird bei zahlreichen Hardware-Implementierungen (z.B. CNAPS 10xx, MA-16[5]) genutzt.

Netzparallelität (Stufe 3)

Die parallele Bearbeitung mehrerer Netze ist sowohl während des Trainings als auch während der Recall-Phase möglich. In beiden Fällen wird jedoch nicht von der inhärenten Parallelität der Netze Gebrauch gemacht, da ihre Ausführung als ein sequentieller Prozeß betrachtet wird, wobei mehrere davon parallel auf einer geeigneten Architektur ausgeführt werden können. Diese Form der Parallelisierung ist daher für einen universellen Parallelrechner geeigneter als für eine direkte VLSI-Implementierung, da der mit dieser Art der Parallelisierung verbundene Verwaltungsaufwand erheblich ist.

[5] siehe hierzu Kapitel 3

2.4 Parallele Hardwarestrukturen für die Beschleunigung neuronaler Netze

Die inhärente Parallelität ist eine Eigenschaft, die neuronale Netze wesentlich von konventionellen Algorithmen unterscheidet und gleichzeitig ein erhebliches Beschleunigungspotential bietet. Hierzu wurde in den vorausgehenden Abschnitten dieses zweiten Kapitels erläutert, welche Parallelitätsebenen existieren, ohne dabei jedoch ein konkretes Neuronenmodell oder eine bestimmte Hardwarearchitektur zu betrachten. In diesem Abschnitt werden nun die aus der Literatur bekannten und für eine VLSI-Implementierung geeigneten Architekturen zur Beschleunigung neuronaler Netze vorgestellt. Auf die zahlreichen Möglichkeiten, massiv parallele Rechner wie z.B. die MasPar MP1, die Connection Machine (CMx) oder die INTEL Paragon für die Simulation neuronaler Netze einzusetzen wird hier nicht eingegangen. An dieser Stelle sei auf die Literatur verwiesen: eine übersichtliche Einführung findet man z.B. bei Serbedzija [Serb96].

2.4.1 Allgemeine Bewertungskriterien paralleler Hardware

Für eine objektive Bewertung und Vergleiche verschiedener Architekturen untereinander ist die Verwendung von Maßzahlen unverzichtbar. Einige für diese Arbeit relevante Bewertungsgrößen sind im folgenden zusammengestellt.

Beschleunigung (Speedup)

Die Beschleunigung b gibt an, wieviel schneller ein Algorithmus oder eine Funktion durch die Optimierung einer Architektur ausgeführt werden kann. Im Hinblick auf die VLSI-Implementierung neuronaler Netze wird diese Optimierung hauptsächlich durch Parallelisierung realisiert werden, die sich im allgemeinen nur auf einen Teil der Gesamtarchitektur auswirken wird. Normiert auf die gesamte Ausführungszeit ist $t_{calc}^{parallel}$ derjenige Anteil, der parallelisiert werden kann. Aus der resultierenden Gesamtbeschleunigung b_{ges} (2.13) kann durch Grenzwertbildung die theoretisch erzielbare Maximalbeschleunigung $b_{ges,max}$ (2.14) ermittelt werden.

$$b_{ges} = \frac{1}{\left(1 - t_{calc}^{parallel}\right) + \frac{t_{calc}^{parallel}}{b_{parallel}}} \qquad (2.13)$$

$$\lim_{b_{parallel} \to \infty} \left(b_{ges}\right) = b_{ges,\max} = \frac{1}{1 - t_{calc}^{parallel}} \qquad (2.14)$$

Diese aus dem Amdahl'schen Gesetz [Amd67] abgeleitete Betrachtung wurde von Wawrzynek [Wawry93] auf Hardware für die Beschleunigung neuronaler Netze angewandt, der die maximal erzielbare Beschleunigung mit

$$b_{\max}^{NN-Hardware} = \frac{1}{Anteil\ aller\ nicht-konnekt.\ Berechnungen} \qquad (2.15)$$

angibt.

Latenzzeit
Die Latenzzeit sagt aus, wie groß die Zeitspanne zwischen dem Start einer Operation, d.h. dem Lesen des ersten Operanden, und dem Ende der Operation, d.h. dem Erscheinen des Ergebnisses am Ausgang eines Rechenwerkes, ist. Die Latenzzeit ist vor allem bei Anwendungen unter Echtzeitanforderungen eine wichtige Größe. Für die folgenden Betrachtungen sei die Latenzzeit auf die Dauer eines Taktes des betrachteten Rechenwerkes normiert (2.16).

$$t_l = \frac{\Delta\tau^{calc}}{T_{Takt}} \qquad (2.16)$$

Datendurchsatz
Neben der Latenzzeit ist der Datendurchsatz eine wichtige Größe zur Beurteilung einer Parallelarchitektur. Im Hinblick auf die Beschleunigung neuronaler Netze wird dieser als Quotient aus der Anzahl verarbeiteter Eingangsdaten eines Netzes und der hierfür benötigten Anzahl von Takten des Rechenwerks betrachtet .

$$d_{NN} = \frac{\#\,Eingangsdaten}{\#\,Takte} \qquad (2.17)$$

Effizienz
Die Effizienz E einer parallelen Architektur drückt das Verhältnis zwischen der Beschleunigung $b_{parallel}$ und der Anzahl eingesetzter Prozessoren k aus, also

$$E = \frac{b_{parallel}}{k}. \qquad (2.18)$$

In einem idealen System besteht ein linearer Zusammenhang zwischen E und k, d.h. jeder weitere Prozessor führt zu einer proportionalen Beschleunigung. Tatsächlich hängt die Effizienz aber davon ab, wie gut sich ein Algorithmus auf eine Architektur abbilden läßt.

2.4.2 Kennzahlen zur Leistungsbewertung neuronaler Hardware

Zur Leistungsbewertung neuronaler Hardware sind viele Maßeinheiten verbreitet, wenngleich sich deren Aussagekraft in Frage stellen läßt [Zell94]. Generell wird zwischen der Trainings- und Recall-Phase unterschieden. In vielen Publikationen wird die Leistung p_{MCPS} der Recall-Phase in MCPS (**M**illion **C**onnections **p**er **S**econd), NIPS (**N**eural **I**nterconnections **p**er **S**econd) oder IPS (**I**nterconnections **p**er **S**econd) angegeben, wobei alle diese Maßzahlen auszudrücken versuchen, wie viele synaptische Verbindungen pro Sekunde berechnet werden (2.19).

$$p_{MCPS} = \frac{alle\ synaptischen\ Verbindungen}{gesamte\ Verarbeitungszeit} \cdot 10^{-6} \quad [MCPS] \qquad (2.19)$$

Analog hierzu gibt es Maßzahlen zur Bewertung des Trainings, welche häufig in der Einheit MCUPS (**M**illion **C**onnection **U**pdates **p**er **S**econd) oder WUPS (**W**eight **U**pdates **p**er **S**econd) angegeben werden. Diese Einheiten sind aber nur dazu geeignet, gleiche Lernverfahren auf verschiedenen Hardwareplattformen miteinander zu vergleichen. Dagegen werden verschiedene Lernverfahren auf der gleichen Hardwareplattform mit Hilfe der Anzahl der Epochen oder Zyklen bis zum Erreichen eines gewünschten Fehlerminimums verglichen, um damit das Konvergenzverhalten zu bewerten. Daß sich trotz solcher, scheinbar eindeutig definierter Maßeinheiten kein objektiver Vergleich verschiedener Hardwareimplementierungen durchführen läßt, soll anhand der folgenden Betrachtungen, die sich hauptsächlich auf die Recall-Phase beziehen, verdeutlicht werden.

Die vielen Parameter, die mit der Implementierung neuronaler Strukturen in Hardware verbunden sind, spiegeln sich kaum in den obigen Leistungsangaben wider. Ein Prozessor, der ausschließlich Gewichte und Aktivitäten mit einer Wortbreite von 8 Bit verarbeitet, erreicht unter Umständen einen wesentlich höheren MCPS-Wert als ein Prozessor, der 16-Bit-Werte verarbeitet und dadurch ein weit größeres Einsatzspektrum abdeckt. In ähnlicher Weise fließt kaum die Flexibilität eines Prozessors in obige Leistungsangaben ein. So kann z.B. ein Parallelprozessor für eine einzige Netztopologie einen sehr hohen MCPS-Wert erreichen, aufgrund seiner unflexiblen Architektur lassen sich andere Netztopologien aber nur sehr ineffizient abbilden. E. Keulen et al. haben daher einige Erweiterungen dieser „klassischen“ Maßzahlen vorgeschlagen, die sich auch auf analoge Implementierungen anwenden lassen. Zur Lösung des Problems unterschiedlicher Wortbreiten wird in [Keulen94] eine neue Maßeinheit MCPPS (**M**illion **C**onnection **P**rimitives **p**er **S**econd) eingeführt, die die Wortbreite beider Operanden ($N_{weights}$ für Gewichte und N_{act} für Aktivitäten) berücksichtigt:

$$p_{MCPPS}^{abs} = N_{weights} \cdot N_{act} \cdot P_{MCPS} \quad \left[MCPS \cdot Bits^2 \right] \qquad (2.20)$$

Abweichend von Keulens Modifikationen bestünde eine Alternative darin, die Operanden auf 16-Bit-Worte zu normieren:

$$p_{MCPPS}^{16Bit} = \frac{N_{weights} \cdot N_{act}}{256\,Bits^2} \cdot p_{MCPS} \quad [MCPS] \qquad (2.21)$$

Diese Leistungsangabe drückt jedoch nur die maximal erzielbare Leistung aus. Insbesondere bei einem Parallelprozessor ist die angegebene Leistung nur dann zu erreichen, wenn alle parallelen Prozeßelemente an der Berechnung beteiligt sind, was jedoch stark von der Netztopologie abhängt. Um eine von der Anwendung weniger stark abhängige Leistungsangabe zu ermöglichen, ist es sinnvoll, die Leistung auf nur eine der parallelen Prozeßeinheiten zu beziehen und in einer separaten Betrachtung die Effizienz einer Architektur bezüglich verschiedener Netztopologien zu untersuchen. Die Leistung, die sich auf nur eine parallele Prozeßeinheit bezieht, wird mit MCPPSPE (**M**illion **C**onnection **P**rimitives **p**er **S**econd / **P**rocessing **E**lement) bezeichnet und wird folgendermaßen definiert:

$$p_{MCPPSPE}^{16Bit} = \frac{N_{weights} \cdot N_{act}}{256\,Bits^2 \cdot k_{PE}} \cdot p_{MCPS} \quad \left[MCPS \,/\, PE \right] \qquad (2.22)$$

2.4.3 Ein Überblick über parallele Architekturen und deren Klassifikation

Seit dem Beginn der Entwicklung paralleler Rechnerarchitekturen in den sechziger Jahren gibt es zahlreiche Entwicklungsrichtungen, die eine Klassifikation nach verschiedensten Kriterien zulassen. Der folgende Überblick soll einen kurzen Abriß über relevante Themengebiete geben und als Einführung in die Parallelisierung neuronaler Strukturen auf der Basis geeigneter Hardwareimplementierungen dienen. Einen umfassenden Überblick über parallele Rechnerarchitekturen findet man z.B. bei R. Duncan [Duncan90] oder bei M.J. Flynn [Flynn95].

Taxonomie nach Flynn

Die am weitesten verbreitete Einteilung paralleler Rechnerstrukturen erfolgt nach der aus dem Jahre 1966 stammenden Taxonomie von Michael J. Flynn [Flynn66]. Hierbei werden Computer nach Befehls- und Datenstrom in vier Klassen eingeteilt:

- SISD: *Single Instruction Stream, Single Data Stream*
- SIMD: *Single Instruction Stream, Multiple Data Streams*
- MISD: *Multiple Instruction Streams, Single Data Stream*
- MIMD: *Multiple Instruction Streams, Multiple Data Streams*

Für die Parallelisierung neuronaler Netze spielen vor allem die Klasse der SIMD- und MIMD-Rechner eine entscheidende Rolle. Es ist zu beachten, daß sich einige andere für die Implementierung neuronaler Netze interessante Architekturen, wie z.B. VLIW[6]-Prozessoren oder systolische Arrays, nicht eindeutig einer dieser Klassen zuordnen lassen und daher die Einbeziehung weiterer Kriterien erfordern [Hennes90].

Speicherarchitektur

Eng verknüpft mit der Entwicklung paralleler Rechnerstrukturen ist diejenige der Speicherarchitekturen. In einer ersten Betrachtung können Parallelrechner in

- Shared Memory und
- Distributed Memory

Maschinen unterteilt werden. Neben dieser Einteilung gibt es viele hybride Systeme, die oftmals über beide Speicherkonzepte verfügen. So ist es vorstellbar, daß die Prozeßelemente eines Parallelprozessors einen Operanden aus einem lokalen Speicher beziehen, während der zweite Operand und die Ergebnisse in einem gemeinsamen Speicher verwaltet werden.

Topologie der Verbindungsnetze

Zur Kommunikation paralleler Prozeßelemente untereinander, mit externem Speichern (Shared Memory) oder zur Kommunikation zwischen Prozessoren werden geeignete Verbindungsstrukturen benötigt. Je nach Speicherarchitektur (verteilt oder zentralisiert), Grad der Parallelität und dem Anwendungsgebiet des Parallelrechners gibt es vielfältige Möglichkeiten der Kommunikation. Generell unterscheidet man zwischen

[6] VLIW steht für **V**ery **L**ong **I**nstruction **W**ord

- statischen und
- dynamischen

Verbindungsnetzen.

Statische Verbindungsnetze zeichnen sich dadurch aus, daß durch eine fest vorgegebene geometrische Anordnung von Prozessoren bzw. Prozeßelementen ein Kommunikationsnetzwerk entsteht. In der einfachsten Form werden Prozeßelemente in einem eindimensionalen Schema angeordnet und mittels sogenannter Broadcast-Busse verbunden. Hierbei übermitteln ein oder mehrere Sender ein Datum an alle anderen Prozeßelemente des Rechners. Dabei unterscheidet man, je nachdem wie viele Sender an der Kommunikation beteiligt sind, zwischen

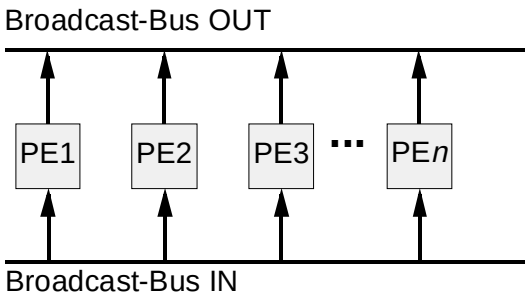


Abbildung 2.7: Beispiel einer Broadcast-Kommunikation

- One-To-All,
- All-To-All und
- K-To-All

Broadcasting [Kumar94]. Im allgemeinen setzt das Broadcasting einen nicht unerheblichen Aufwand für das Routing in den Prozeßelementen voraus, um Daten innerhalb einer beliebigen Struktur (Torus, Hypercube) zu versenden. Als Ausnahme sind hier eindimensionale Busarchitekturen (Abbildung 2.7) zu erwähnen, bei welchen ein Routing in den Prozeßelementen entfällt. Hierdurch ergeben sich von der Anzahl paralleler Prozeßelemente weitgehend unabhängige Transferzeiten.

Eine weitere Möglichkeit zur statischen Verbindung paralleler Prozeßelemente besteht im Aufbau ein- und zweidimensionaler Pipelinestrukturen in Form sogenannter systolischer Arrays. Grundsätzlich unterscheidet man drei Varianten:

- systolische Kette (systolic chain),
- systolischer Ring (systolic ring) und
- planares systolisches Feld (systolic mesh).

In vielen Bereichen der digitalen Signalverarbeitung und der Simulation neuronaler Netze findet man diese Architekturen, die im folgenden Kapitel 2.4.4 genauer betrachtet werden.

Statische Verbindungsnetze zeichnen sich im allgemeinen durch eine geringe Flexibilität aus. Dies betrifft sowohl die Kommunikation der Prozeßelemente untereinander als auch die Kommunikation zwischen Prozeßelementen und Speicher. Dynamische Verbindungsnetze überwinden diese Problematik, indem sie beliebige Punkt-zu-Punkt-Verbindungen ermöglichen, ohne daß für die Anordnung der Prozeßelemente selbst Einschränkungen gelten. Am häufigsten werden dynamische Verbindungsnetze für die Kommunikation zwischen Prozeßelementen und Speichereinheiten verwendet.

Zur gleichzeitigen Kommunikation (non blocking) von k Prozeßelementen mit k verteilten Speichern eignen sich sogenannte Crossbars. Ihre Kosten von $O(k^2)$ sind aber nur bei kleinen Werten von k vertretbar, weil allzu schnell die Verdrahtung eines Crossbars mehr Fläche in Anspruch nimmt als die Prozeßelemente selbst. Beim digitalen Signalprozessor TMS320C80 [Texas96] von Texas Instruments kommt z.B. ein solcher Crossbar für die Verbindung von vier Prozeßelementen mit vier Speicherblöcken zum Einsatz. Während Busarchitekturen (Broadcasting) sich in bezug auf die Kosten sehr gut skalieren lassen, nimmt mit jeder weiteren Prozeßeinheit an diesem Bus die Leistung ab, da immer nur ein Teilnehmer auf den Bus schreiben kann. Crossbars lassen sich in bezug auf die Leistung hervorragend skalieren, da sich die parallelen Verbindungen gegenseitig nicht blockieren können (non blocking), mit jedem weiteren Teilnehmer nehmen aber die Kosten quadratisch zu. Einen Mittelweg stellen sogenannte Multistage-Netzwerke dar. Der bekannteste Vertreter dieser Klasse von Verbindungsnetzen ist das Omega-Netzwerk [Lawrie75], bei dem k Schaltblöcke mit je $log_2(k)$ Ein- und Ausgängen hintereinander angeordnet sind.

2.4.4 Systolische Arrays

Die Grundidee systolischer Arrays stammt aus den siebziger Jahren und wurde erstmals von H. T. Kung und C. E. Leiserson [KungHT78] veröffentlicht. Motivation für die Entwicklung systolischer Arrays war der Wunsch, geeignete Algorithmen direkt in Hardware zu implementieren, eine Balancierung von Kommunikations- und Rechenaufwand zu erzielen und die Algorithmen soweit wie möglich zu parallelisieren. Ziel war es, ein ‘Load Balancing’ zu erreichen, d.h. eine gleichmäßige Auslastung paralleler Prozessoren anzustreben [KungHT80]. Eine exakte Definition des systolischen Arrays ist in der Literatur nicht zu finden, es gibt jedoch z.B. bei Kumar et al. [Kumar94] eine hilfreiche Erklärung:

> *„In a systolic system, data elements flow from external memory in a rhythmic fashion, passing through many cells before the results return to external memory, much as blood circulates to and from the heart.“*

Anhand einiger Eigenschaften, die in der folgenden Tabelle zusammengefaßt sind, charakterisiert Viredaz [Viredaz94] systolische Arrays folgendermaßen:

- Modularität — *das Array ist aus identischen Prozeßelementen aufgebaut, die in einer regulären Struktur angeordnet werden.*
- Synchrone Verarbeitung — *alle Operationen sind mit einer Clock synchronisiert*
- Lokale Kommunikation — *Prozeßelemente können nur über lokale Verbindungen zu direkten Nachbarn Daten austauschen; Zugriffe auf externe Speicher sind nur an den Array-Grenzen möglich*

Die Eignung systolischer Arrays für die Beschleunigung neuronaler Netze mit Hilfe optimierter VLSI-Strukturen wurde von S.Y. Kung [KungSY89, KungSY91, KungSY93] vorgeschlagen, der sich jedoch weitgehend auf Netze mit Multiply-Accumulate-Funktion wie z.B. Multilayer-Perzeptronen oder rückgekoppelte Backpropagation-Netze konzentrierte. Obwohl es keine genaue Definition gibt, haben sich in der Literatur drei Formen systolischer Arrays etabliert: die Kette, der Ring und das planare Feld.

Systolische Kette

Die einfachste Form eines systolischen Arrays ist eine systolische Kette (Abbildung 2.8), die sich durch eine eindimensionale Anordnung gleichartiger Prozeßelemente auszeichnet. Damit läßt sich die Matrix-Vektor-Multiplikation, die z.B. Bestandteil von Multilayer-Perzeptronen ist, durch Pipelining beschleunigen.

Betrachtet wird folgende Operation:

$$\underline{a} = \underline{W} \cdot \underline{o} \qquad (2.23)$$

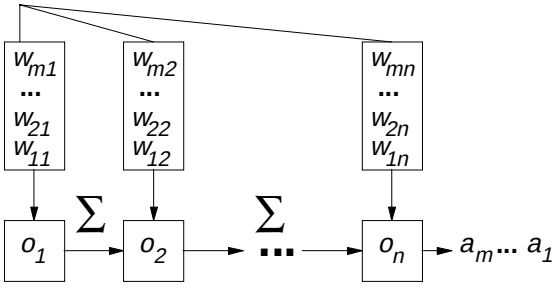
bzw.

$$\left(a_1, a_2, ..., a_m\right)^T = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} \cdot \left(o_1, o_2, ..., o_n\right)^T \qquad (2.24)$$

Zur Abbildung dieser Operation auf eine systolische Kette schlägt Kung [KungSY93] vor, die Operation zunächst in einem Datenabhängigkeitsgraphen (DG) darzustellen und ausgehend von diesem eine geeignete Abbildung zu definieren. Im Falle obiger Operation existieren zwei Möglichkeiten, diese auf eine systolische Kette abzubilden (Abbildung 2.8).

Variante (a) :
n parallele Prozeßelemente

lokale Speicher:
Spaltenvektoren der Gewichtsmatrix



Variante (b) :
m parallele Prozeßelemente

lokale Speicher:
Zeilenvektoren der Gewichtsmatrix

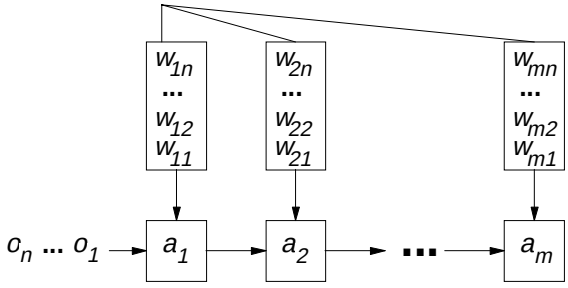


Abbildung 2.8: Zwei Varianten einer systolischen Kette

Bei der ersten Variante (a) sind die n Elemente des Eingangsvektors $\underline{o}$ in n parallelen Prozeßelementen gespeichert und werden mit jeweils m Gewichten multipliziert. Die Produkte werden nach rechts zum nächsten Prozeßelement gesendet und dort zum entsprechenden Produkt addiert. Die Endsummen strömen dann am rechten Ende der Kette nach außen.

Bei der zweiten Variante (b) durchströmen alle n Elemente des Vektors $\underline{o}$ von einer Seite alle Prozeßelemente und werden dort mit den entsprechenden Gewichten multipliziert und die Produkte akkumuliert, so daß in jedem der m Prozeßelemente ein Element des Ergebnisvektors $\underline{a}$ berechnet wird.

Systolischer Ring

Ausgehend von einer systolischen Kette entsteht ein systolischer Ring durch Verbinden des ersten mit dem letzten Prozeßelement. Mit dieser Torusstruktur läßt sich wie schon bei der systolischen Kette die Multiplikation einer Matrix $\underline{W}(m \text{ x } n)$ mit einem Vektor $\underline{o}(n \text{ x } 1)$ parallelisieren. Prinzipiell sind auch hier beide Varianten der systolischen Kette (n bzw. m parallele Prozeßelemente) möglich. Im Fall der Varianten (b) (m_0 parallele Prozeßelemente) hat ein Ring gegenüber einer Kette den Vorteil, daß sich die Abbildung einer Schicht mit $m > m_0$ Neuronen auf die Hardware (Partitionierung[7]) leichter vornehmen läßt. Hierzu rotieren die Partialsummen mehrfach in der Torusstruktur und werden anstatt in einem Umlauf mit m Prozeßelementen in S Umläufen mit

$$m_0 = \frac{m}{S} \qquad (2.25)$$

parallelen Prozeßelementen berechnet [KungSY89, KungSY91] (Abbildung 2.9).

Neben dieser hardwarefreundlichen Partitionierung zeichnen sich systolische Torusstrukturen dadurch aus, daß sie sich für die Implementierung mehrschichtiger Feedforward-Netze eignen. Wird nach Abschluß eines Zyklus die Transferfunktion f auf den Ergebnisvektor $\underline{a}$ angewendet, so können diese Werte im Torus verbleiben und dienen für die Berechnung der folgenden Schicht als Eingangswerte. Hierdurch entfällt das Leeren und erneute Laden der Pipelinestruktur. Der gleiche Effekt kann zum Training rückgekoppelter Backpropagation-Netze genutzt werden [Jones91].

Die folgende Abbildung zeigt beispielhaft die Matrix-Vektor-Multiplikation $\underline{W}(4 \text{ x } 8) \cdot \underline{o}(8 \text{ x } 1)$ auf einem systolischen Ring mit lediglich vier Prozeßelementen. Die Partialsummen rotieren zweimal im Torus. Die Latenzzeit von einem Takt je Prozeßelement spiegelt sich in der Reihenfolge der Gewichte wider, welche zeilenweise, um jeweils eine Spalte rotiert, in den lokalen Speichern der Prozeßelemente abgelegt sind.

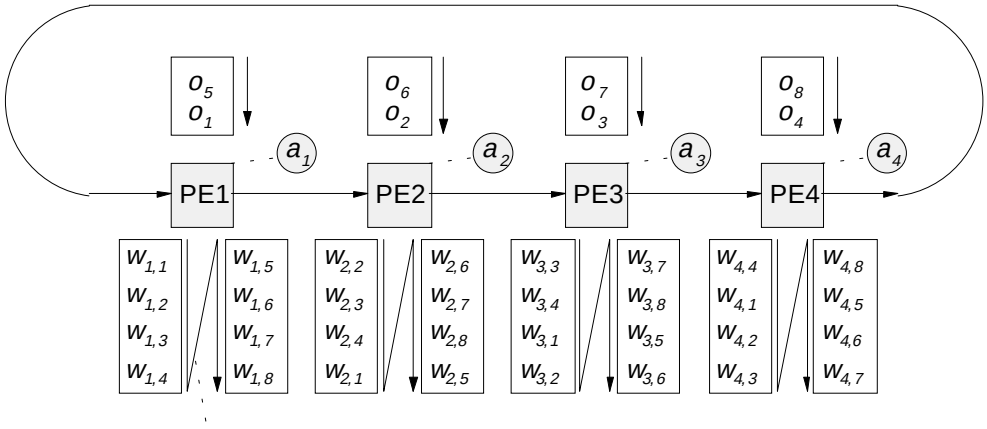


Abbildung 2.9: Matrix (4 x8) - Vektor (8 x 1) Multiplikation auf einem systolischen Ring mit 4 PEs

[7] Siehe hierzu auch Kapitel 4.3.5

Planare systolische Felder

Die dritte Form systolischer Arrays sind die sogenannten planaren systolischen Felder (*systolic mesh)*. Sie zeichnen sich dadurch aus, daß ihre Prozeßelemente in einem zweidimensionalen Array angeordnet und die Elemente nur mit ihren direkten Nachbarn verbunden sind (Abbildung 2.10). Sie eignen sich insbesondere zur Multiplikation zweier Matrizen $\underline{W}(m \text{ x } n) \cdot \underline{O}(n \text{ x } k)$, wie es bei Feedforward-Netzen der Fall ist, wenn k Aktivitätsvektoren $\underline{o}(n \text{ x } 1)$ zu einer Matrix $\underline{O}(n \text{ x } k)$ zusammengefaßt werden. Wie schon bei den zuvor vorgestellten Varianten systolischer Arrays, existieren auch hier zwei Möglichkeiten, diese Operation auf die entsprechende Hardware abzubilden. In der folgenden Abbildung 2.10 sind beide Varianten am Beispiel einer (2 x 2) · (2 x 2)-Multiplikation dargestellt.

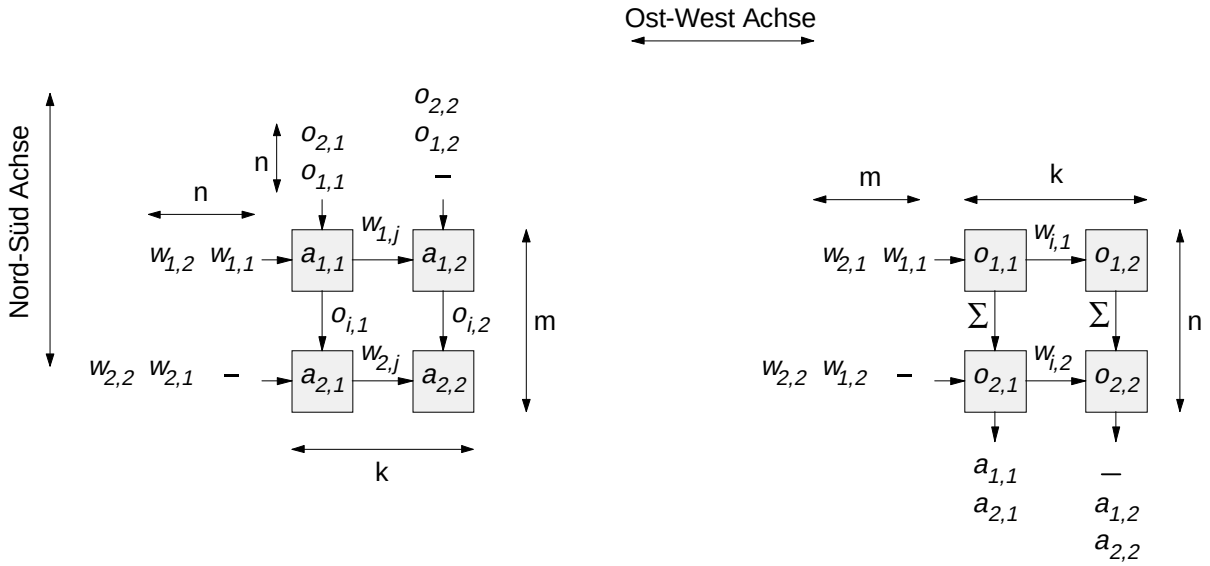


Abbildung 2.10: Zwei Varianten eines planaren systolischen Feldes

Bei der ersten Variante strömen beide Operanden ($\underline{W}$ und $\underline{O}$) in das systolische Feld, dessen Prozeßelemente nach n Schritten jeweils ein Element der Ergebnismatrix $\underline{A}$ berechnen. Die Anzahl der Zeilen (Ost-West-Achse) entspricht der Dimension m, d.h. der Anzahl der Neuronen in der betrachteten Schicht; die Anzahl der Spalten (Nord-Süd-Achse) entspricht der Anzahl der berechneten Muster (k). Entlang der Ost-West-Verbindungen werden die Gewichte, entlang der Nord-Süd-Verbindungen die Aktivitäten durch das systolische Array gereicht.

Bei der zweiten Variante werden die Aktivitätswerte in den Prozeßelementen gespeichert, und die Gewichte strömen beispielsweise von Westen in das Feld, welches n Zeilen (Anzahl der Neuronen in der vorherigen Schicht) und k Spalten (Anzahl der berechneten Muster) besitzt. Entlang der Nord-Süd-Verbindungen werden die Partialsummen, entlang der Ost-West-Verbindungen die Gewichte transportiert. Am unteren Ende strömen die Elemente der Ergebnismatrix $\underline{A}$ aus dem Array.

Neben diesen Basisarchitekturen gibt es mittlerweile zahlreiche Abwandlungen [Leh93], die vom Prinzip her aber alle auf diesen Architekturen aufbauen.

2.5 Zusammenfassung

Dieser einführende Überblick hat anhand einer Analyse (Abbildung 2.1) von 154 Anwendungen mit neuronalen Netzen gezeigt, daß Multilayer-Perzeptron-Netze und Kohonen-Netze die wichtigste Rolle in der Praxis spielen. Da RBF-Netze mathematisch gesehen auf den Funktionen der beiden erstgenannten Netztypen aufbauen, lassen sich diese ohne wesentlichen Zusatzaufwand mit einer Hardware für MLP- und Kohonen-Netze beschleunigen und werden daher Gegenstand weiterer Betrachtungen sein. Da die Struktur der Steuerung für rückgekoppelte Netze wesentlich von der für vorwärtsgerichtete Netze abweicht, werden diese aus den weiteren Betrachtungen ausgeklammert.

Zur Beschleunigung dieser Netze unter Nutzung verschiedener Parallelitätsebenen (Kapitel 2.3) wurden verschiedene aus der Literatur bekannte Architekturen vorgestellt. Neben eindimensionalen Bus-Strukturen eignen sich vor allem systolische Arrays für eine VLSI-Implementierung (Kapitel 2.4). Bei der Definition eines Maßes zur Bewertung von Hardware für neuronale Netze hat sich gezeigt, daß es äußerst schwierig ist, sämtliche Aspekte zu berücksichtigen. Für die in dieser Arbeit geltenden Rahmenbedingungen (digitale Implementierung, Festkommaarithmetik) eignet sich vor allem die Maßeinheit p_{MCPPS}^{16Bit}, die einen Leistungsvergleich verschiedener Architekturen, normiert auf 16-Bit-Operanden, erlaubt.

Das folgende dritte Kapitel gibt einen kurzen Überblick über den Stand der Technik bei digitaler Hardware zur Beschleunigung neuronaler Netze, wobei die im zweiten Kapitel dargestellten Kriterien zur Klassifikation und Bewertung neuronaler Hardware verwendet werden.

3 Stand der Technik bei digitaler Hardware für neuronale Netze

Bei der Entwicklung neuronaler Hardware hat sich in den vergangenen Jahren eine große Diversifikation abgezeichnet, die sich nicht zuletzt in einer gestiegenen Anzahl kommerziell erhältlicher neuronaler Hardwarekomponenten widerspiegelt. Aber auch im Bereich der Forschung gibt es immer vielfältigere Implementierungen, die nicht alle in erster Linie der Beschleunigung neuronaler Operationen dienen, sondern eher als Machbarkeitsstudien neuer, häufig biologisch inspirierter Neuronenmodelle zu sehen sind. Die folgende Übersicht wird sich daher auf solche digitale Hardwarekomponenten beziehen, die vorrangig der Beschleunigung neuronaler Operationen dienen, eine höhere Leistung als heutige Standardprozessoren bieten (> 50 MCPS), eine hohe Integrationsdichte besitzen (maximal zwei Chips - ein paralleles Rechenwerk und ein Steuerwerk) und kommerziell verfügbar sind.

Im ersten Teil dieses Kapitels werden solche Systeme betrachtet, die aufgrund ihrer Flexibilität als General-Purpose-Prozessoren eingesetzt werden und durch parallele Architekturkomponenten die Beschleunigung neuronaler Operationen ermöglichen. Im zweiten Teil werden dann anwendungsspezifische Parallelprozessoren betrachtet, die aufgrund ihrer Architektur auf wenige Netzwerkmodelle beschränkt sind (single model). Der dritte Teil stellt dann schließlich Vertreter der sogenannten General-Purpose-Neuro-Prozessoren vor, die auf der Basis einer parallelen Architektur eine Vielzahl neuronaler Netze beschleunigen können (multi model).

Wenngleich die folgende Aufstellung keinen Anspruch auf Vollständigkeit erhebt, soll sie zumindest einen groben Überblick über den Stand der Technik bei digitaler Hardware zur Beschleunigung neuronaler Netze verschaffen. Neben einer kurzen Betrachtung der Architektur wird versucht, folgende Fragestellungen zu beantworten.

1. *Wie flexibel ist die Implementierung, d.h. welche Algorithmen lassen sich beschleunigen und wie effizient wird dabei die parallele Hardware genutzt?*
2. *Wie stellt sich die Schnittstelle zum Benutzer dar?*
3. *Wie kosteneffektiv ist die Implementierung?*

Viele der hier vorgestellten Hardwarekomponenten zur Beschleunigung neuronaler Netze wurden bereits in zahlreichen Veröffentlichungen präsentiert, und ihre Leistung wurde analysiert. Darüber hinaus findet man umfangreiche Zusammenfassungen über neuronale Hardwarekomponenten bei P. Ienne [Ienne95a], J. Heemskerk [Heemsk95], C. Lindsey [Lindsey94], M. Glesner [Glesner94] und im World Wide Web [Cern98].

3.1 Parallele Systeme basierend auf digitalen General-Purpose-Prozessoren

Als der Entwurf anwendungsspezifischer Prozessoren mangels geeigneter Technologien und Entwicklungswerkzeugen noch verhältnismäßig teuer und aufwendig war, entwickelten zahlreiche Forschergruppen auf der Basis von Standardprozessoren Parallelrechner zur Beschleunigung neuronaler Netze. Ein Vertreter dieser Gruppe ist der RAP (**R**ing **A**rray **P**rocessor) des International Computer Science Institute, Berkeley, der aus maximal 16 in einem Ring angeordneten Karten mit je vier digitalen Signalprozessoren vom Typ TMS320C30 besteht [Morgan90, Beck90]. Die maximale Leistung dieses Systems mit 64

Prozessoren wird mit 56,9 MCPS bzw. 12,2 MCUPS angegeben, als Kommunikationselemente wurden Xilinx-FPGAs verwendet. Eine vergleichbare Architektur besitzt auch der Parallelrechner MUSIC (**Mu**ltiprocessor **S**ystem with **I**ntelligent **C**ommunication) der ETH Zürich [MüllerU95], der für die ringförmige Kommunikationsstruktur Transputer vom TypT805 verwendet. Die insgesamt 60 parallelen Prozeßelemente, digitale Signalprozessoren von Motorola (DSP96002), ermöglichen eine maximale Leistung von 330 MCPS, wobei es sich hierbei um Fließkommaoperationen handelt. Das gesamte System hat eine Leistungsaufnahme von 800 Watt.

Wenngleich beide Systeme auf Standardprozessoren aufbauen, so lassen ihre Größe und ihre Komplexität kaum eine einfache Handhabung und günstige Herstellkosten zu. Einen völlig anderen Weg beschreiten die beiden folgenden Systeme (Kapitel 3.1.1 und 3.1.2). In beiden Fällen wurden General-Purpose-Prozessoren um parallele Architekturmerkmale erweitert.

3.1.1 Standardprozessoren mit Multimedia-Erweiterungen

Im vergangenen Jahr haben viele namhafte Hersteller wie z.B. AMD [AMD98] und INTEL [Intel98a] ihre General-Purpose-Prozessoren um sogenannte Multimedia-Funktionen erweitert. Dabei handelt es sich um eine schnelle Integer-Arithmetik, im Byte-, 16-Bit- bzw. 32-Bit-Format zur parallelen Ausführung von Filterungsalgorithmen im Audio- und Bildverarbeitungsbereich. Aufgrund der mathematischen Verwandtschaft dieser Filterungsalgorithmen mit neuronalen Netzen sind diese Erweiterungen auch zur Beschleunigung neuronaler Operationen geeignet. Stellvertretend wird im folgenden die MMX (**M**ulti**m**edia E**x**tensions)-Erweiterung von INTEL [Intel98a, CT97, Jaehne97] vorgestellt.

MMX-Architekturerweiterung von INTEL

Kernstück der MMX-Architekturerweiterung sind zwei parallele Pipelines, genannt MMX-U und MMX-V, die unmittelbaren Zugriff auf den L1-Datencache besitzen und eine gleichzeitige Verarbeitung von jeweils zwei 64-Bit-Operanden erlauben. Mit MMX wurde neben einer Verdopplung der beiden Caches (Daten- und Programm-Cache) auf jeweils 16 KBytes der neue Datentyp PACKED eingeführt. Hierbei handelt es sich um ein Datenformat, bei welchem acht Bytes, vier 16-Bit-Worte oder zwei 32-Bit-Worte als ein 64-Bit-Wort aufgefaßt, d.h. in dieses 64-Bit-Format gepackt werden. Je nach gewähltem Format werden dann innerhalb einer Pipeline bis zu acht Operanden parallel verarbeitet. Nach Flynn's Taxonomie entspricht die zugrunde liegende Architektur einem SIMD-Prozessorfeld. Insgesamt stehen 57 zusätzliche OP-Codes und acht 64-Bit-(MMX) Register zur Verfügung. Für die Beschleunigung neuronaler Netze, die auf der Multiply-Add-Operation aufbauen, ist vor allem der OP-Code PMADDWD (**P**acked**M**ultiply**AddW**or**d**) von Interesse, da dieser die Multiplikation zweier 16-Bit-Operanden und anschließende Addition innerhalb von drei Takten erlaubt. Da die Operation in einer Pipeline abläuft, können aber trotzdem in jedem Takt zwei neue Operanden eingelesen werden. Voraussetzung ist dabei, daß sämtliche Operanden in MMX-Registern zur Verfügung stehen bzw. die Summen dort abgelegt werden. Zur Konvertierung der verschiedenen Datenformate (8, 16, 32 und 64 Bit) untereinander stehen verschiedene Packbefehle und Shift-Operationen zur Verfügung. Obwohl die beiden MMX-Pipelines U und V jeweils zwei 64-Bit-Operanden verarbeiten können, ist es dennoch nicht möglich, acht (16 Bit) MAC-Operationen simultan auszuführen. Dies liegt daran, daß sich die beiden Pipelines einen (32 x 32 Bit)-Multiplizierer teilen. Somit können lediglich vier (16 x 16 Bit)-

Multiplikationen gleichzeitig ausgeführt werden. Ein stark vereinfachtes Blockdiagramm der MMX-Erweiterung ist in der Abbildung 3.1 dargestellt. Aktuelle Informationen, Applikationsbeispiele und Datenblätter findet man bei INTEL [Intel98a].

Die Erweiterung der INTEL x86-Architektur um eine schnelle, parallele Integer-Arithmetik eröffnet mit Sicherheit für viele Anwendungen aus dem Multimedia-Bereich ein erhebliches Beschleunigungspotential. Inwieweit dies auf neuronale Netze zutrifft, wird im Kapitel 6.2 dieser Arbeit anhand von Benchmarks genauer untersucht. Aus Sicht des Programmierers besitzt die Programmierung der MMX-Unit eines INTEL-Prozessors einige Nachteile. Aus Kompatibilitätsgründen werden MMX-Register wie Register der Fließkommaeinheit behandelt. Dies hat zur Folge, daß innerhalb einer Task entweder nur MMX- oder nur Fließkommaoperationen ausgeführt werden können. Bei einem Task-Wechsel müssen dann die Inhalte der Register in den externen Speicher ausgelagert bzw. von dort geladen werden. Ein zweiter erheblicher Nachteil ist die mangelnde Unterstützung seitens der Compilerhersteller. MMX kann momentan nur in Assembler programmiert werden - dies hat eine nicht zu vernachlässigende Einarbeitungszeit zur Folge, da zusätzlich zu den Schwierigkeiten der Programmierung in Assembler noch Besonderheiten der Parallelprogrammierung kommen. Dies gilt insbesondere dann, wenn es darum geht, einen Algorithmus optimal auf die MMX-Architektur abzubilden. Ein dritter zu erwähnender Nachteil ist die Ankopplung der MMX-Pipelines an den externen Speicher. Die zweistufige Cache-Struktur (L1-Cache auf dem Chip, L2-Cache außerhalb) erfordert eine überlappende Verarbeitung mehrerer Datensätze, wofür die acht MMX-Register nicht ausreichen. Hierdurch sind Speicherzugriffe anstelle von Registerzugriffen erforderlich, welche die Leistung erheblich mindern (Kapitel 6.2.1).

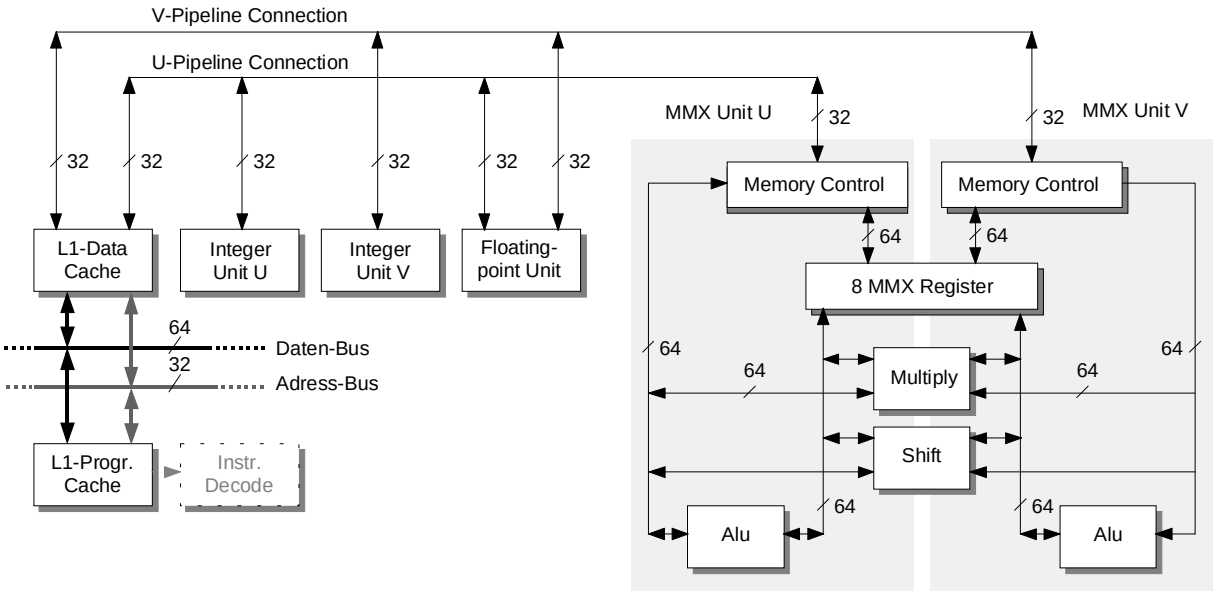


Abbildung 3.1: Vereinfachtes Blockdiagramm der INTEL-MMX-Architektur-Erweiterung

3.1.2 Digitale Signalprozessoren

TMS320C62x DSP von Texas Instruments

Der im Jahre 1997 vorgestellte digitale Signalprozessor TMS320C62x von Texas Instruments [Texas97a] stellt mit einer Maximalleistung von 1600 MIPS (**M**illionen **I**nstruktionen **p**ro **S**ekunde) bei einem Takt von 200 MHz die Spitze derzeit verfügbarer Signalprozessoren mit Integer-Arithmetik dar. Mit zwei identischen Funktions- und Registerblöcken, die weitgehend unabhängig voneinander arbeiten, eignet sich der C62x für vektor- und matrixbasierte Operationen und scheint daher nicht nur für die klassische Signalverarbeitung, sondern auch für die Beschleunigung neuronaler Operationen geeignet zu sein. Aufgrund der vielfältigen Einsatzmöglichkeiten solcher digitaler Signalprozessoren (mobile Kommunikation, Computernetzwerke, Satellitennavigation etc.) ist mit einem schnellen Preisverfall zu rechnen, so daß dieser DSP als eine sehr preiswerte Alternative zu einem digitalen Neuro-Chip betrachtet werden kann. Im Gegensatz zum älteren TMS320C80 [Texas96], der vier parallele DSPs auf einem Chip vereint und damit einen MIMD-Rechner darstellt, handelt es sich beim C62x (Abbildung 3.2) um einen VLIW-Prozessor, dessen parallele Prozeßelemente durch ein (256-Bit) Befehlswort synchron gesteuert werden. Neben dem C6200-CPU-Core befinden sich je 64 KBytes Programm- und Datenspeicher auf dem Chip. Für große Programme, deren Code außerhalb des DSPs verwaltet werden muß, kann ein Teil des internen Programmspeichers als Programm-Cache genutzt werden. Neben einer Steuereinheit für DMA-Zugriffe befinden sich weitere Peripheriekomponenten auf dem Chip, wie z.B. zwei unabhängige Timer sowie zwei schnelle serielle Ports, die verschiedene, im Telekommunikationsbereich übliche Protokolle unterstützen.

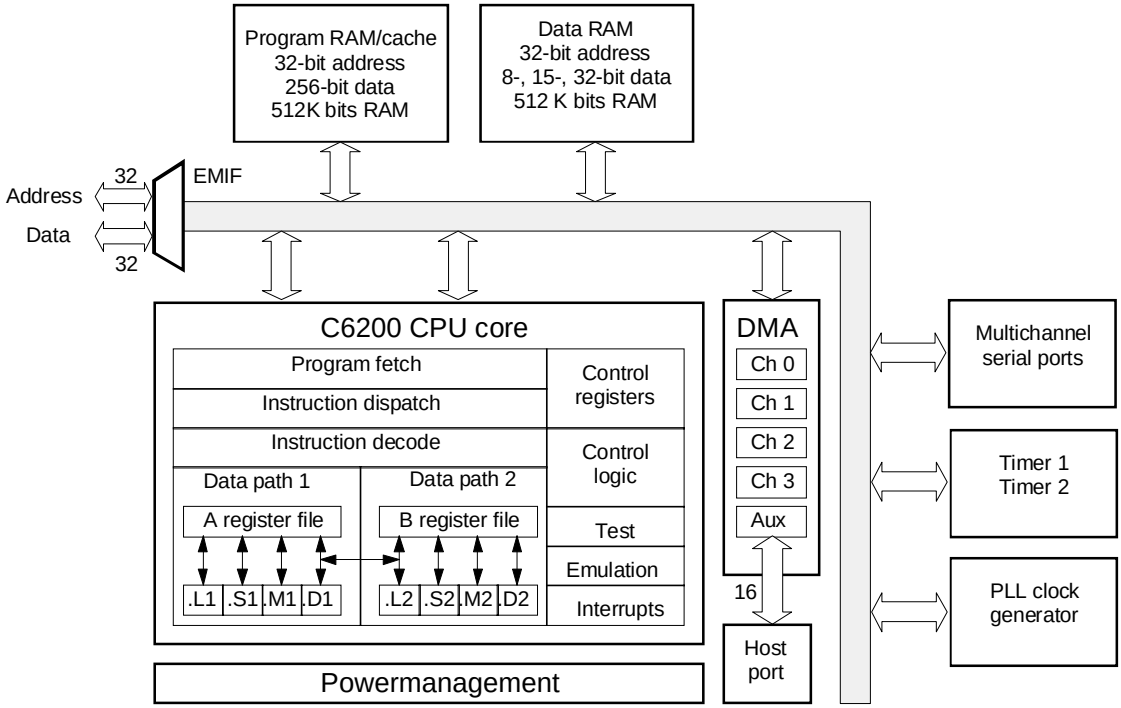


Abbildung 3.2: Blockdiagramm des TMS320C62x-Signalprozessors von Texas Instruments

Für die Beschleunigung neuronaler Netze ist insbesondere die Architektur des C6200 CPU Cores, die Anbindung an in- und externe Speicher sowie dessen Programmierung von Interesse.

Jeder der beiden Data-Path-Blöcke in Abbildung 3.2 enthält jeweils vier funktionelle Einheiten (L,S,M und D) und ein sogenanntes Registerfile, bestehend aus 16 32-Bit-Registern. Für die Parallelisierung von Multilayer-Perzeptronen ist vor allem die Anzahl parallel ausführbarer Multiply-Accumulate-Funktionen wichtig. Die L-, S-, und D-Einheiten können zwar parallel auf verschiedenen Operanden Additionen durchführen, doch steht pro Data-Path-Einheit nur ein Multiplizierer (M-Einheit) zur Verfügung, so daß insgesamt pro Takt zwei (16 x 16 Bit)-MAC-Instruktionen ausgeführt werden können. Aufgrund von Latenzzeiten und einer Pipelinestruktur des Multiplizierers kann der C62x effektiv in zwei Takten vier Multiplikationen und Additionen ausführen. Dabei handelt es sich nicht um eine kurzzeitige Spitzenleistung. Durch die Möglichkeit, während der Ausführung einer Operation gleichzeitig Daten vom internen Datenspeicher des C62x in beide Registerfiles nachzuladen, kann auch während längerer Akkumulationszyklen die angegebene Leistung erreicht werden. Der interne Speicher ist in vier Sektionen mit insgesamt 8192 16-Bit-Blöcken unterteilt, so daß, verschiedene Adressen vorausgesetzt, gleichzeitig zwei Lese- oder Schreiboperationen erfolgen können. Dies ist für die Parallelisierung neuronaler Netze unbedingt erforderlich, da der C62x nur einen gemeinsamen Speicher für Gewichte und Aktivitäten besitzt. Da nicht nur die CPU Zugriff auf den internen Speicher hat, sondern auch per DMA Daten eingelesen werden können, verwaltet ein Memory-Controller alle Zugriffe auf den internen Speicher. Dieser führt auch die Adreßdekodierung durch und erlaubt hierdurch einen transparenten Zugriff auf Peripherieeinheiten und externe Speicher.

Zur Programmierung des C62x stehen umfangreiche Entwicklungswerkzeuge (C-Compiler, Assembler, Debugger, Profiler) zur Verfügung. Die optimale Abbildung von Algorithmen auf die parallele Hardware soll einerseits durch einen optimierenden Compiler und andererseits durch die Einbindung sogenannter Intrinsic-Functions in den C-Code unterstützt werden. Hierbei handelt es sich um parametrisierbare Assembler-Funktionen, die z.B. eine schnelle MAC-Funktion realisieren.

Basierend auf der hohen Taktfrequenz von 160 bzw. 200 MHz und der Möglichkeit, sowohl Gewichte als auch Aktivitäten im 16-Bit-Format zu verarbeiten, läßt der C62x eine hohe Leistung vermuten. Um hier Klarheit zu schaffen, wurden umfangreiche Benchmarks durchgeführt, die im Kapitel 6.2.2 ausführlich diskutiert werden.

3.2 Digitale Single Model Neuro-Chips

Single Model Neuro-Chips zeichnen sich dadurch aus, daß sie im allgemeinen nur ein Netzwerk bzw. Neuronenmodell unterstützen und infolgedessen eine sehr einfache Handhabung besitzen. Von den vielen aus der Literatur bekannten Chips werden im folgenden einige typische Vertreter vorgestellt.

SIOP2 Neuro-Chip von IMS Stuttgart

Der SIOP2 (**S**erial **I**nput **O**peration **P**arallel) Neuro-Chip [Neusser96, Friebe97] des IMS Stuttgart stellt eine vollständig parallele Implementierung von Multilayer-Perzeptronen und rückgekoppelten Netzen dar, d.h. entsprechend der Analyse von Kapitel 2.3 parallelisiert SIOP2 neuronale Netze auf Verbindungsebene. Sowohl die Gewichte als auch die Aktivierungsfunktion wurden beim SIOP2 auf dem Chip integriert. Um den Verdrahtungsaufwand infolge der vollständigen Parallelisierung in Grenzen zu halten, liest und verarbeitet SIOP2 Aktivitäten bitseriell. Jede Synapse besteht aus einem Gewichtsspeicher und einem Seriell-Parallel-Multiplizierer. Sämtliche Synapsenausgänge werden mit Hilfe eines seriellen Addierers zu einer Aktivierung aufsummiert, die anschließend in der Neuronen-Basiszelle zur Berechnung der Ausgangsaktivität dient. Anstelle der häufig verwendeten logistischen Aktivierungsfunktion (Abbildung 2.3) wurde bei SIOP2 eine quadratische Kennlinie (3.1) implementiert, die sich relativ einfach in Hardware implementieren läßt.

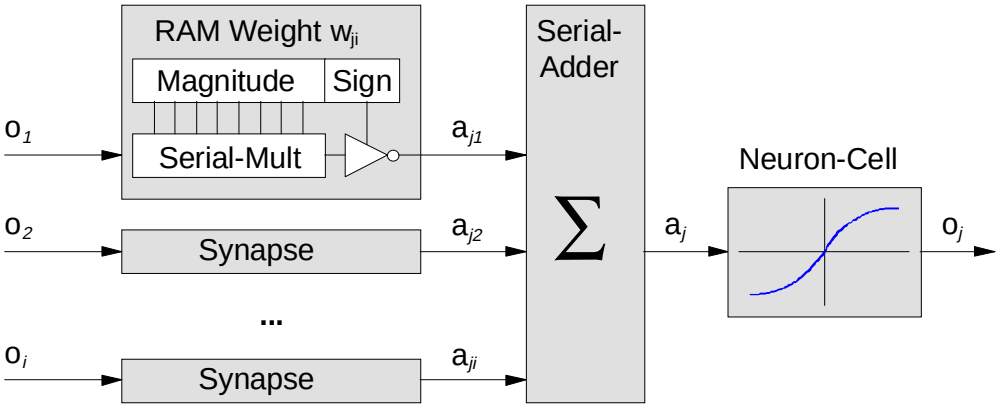


Abbildung 3.3: Bitserielle Architektur des SIOP2 (maximal 40 Eingangsneuronen, maximal 40 MHz Taktfrequenz)

$$o_j(t+1) = \begin{cases} \mathrm{sgn}\left(a_j\right) \cdot 2 \cdot \left|a_j\right| \cdot \left(1 - \frac{a_j}{2}\right) & \left|a_j\right| < 1 \\ \mathrm{sgn}\left(a_j\right) & \left|a_j\right| \geq 1 \end{cases} \qquad (3.1)$$

Die maximale Leistung des SIOP2 beträgt bei 16 Bit Aktivitäten und 10 Bit Gewichten 350 MCPS. Wird dieser Wert entsprechend Gleichung (2.21) auf 16-Bit-Operanden normiert, ergibt sich eine Maximalleistung von 219 MCPS. Die hohe Integrationsdichte und die hohe Verarbeitungsleistung sind eindeutige Vorteile dieses Neuro-Chips. Daher ist er vor allem für Anwendungen im Bereich der Automatisierungstechnik attraktiv, da nur wenige zusätzliche Peripheriekomponenten benötigt werden. Die geringe Flexibilität des SIOP2 - es werden nur 11 Netzkonfigurationen unterstützt - schränken jedoch seine Einsatzmöglichkeiten als universelles Hardware-Beschleunigungswerkzeug erheblich ein.

NM6403 von Module Research, Moskau

Beim NM6403 von Module Research [Vixne96] handelt es sich um einen digitalen superskalaren Neuroprozessor, dessen Operanden, ähnlich wie beim L-Neuro von Philips [Mauduit92], eine variable Länge besitzen können. Im Gegensatz zum L-Neuro verarbeitet der NM6403 jedoch alle Daten mit einer variablen Breite zwischen 1 und 64 Bit wortparallel. Der NM6403 unterstützt ausschließlich vorwärtsgerichtete, vollvernetzte Strukturen ohne Rückkopplungen, bei denen ein Neuron die bereits in Kapitel 2.2 erläuterte Funktion (2.1), (2.2) realisiert. Als Transferfunktion sind lediglich eine Schwellwert- und eine lineare Sättigungsfunktion möglich, die beide auf dem Chip implementiert sind.

Neben einem für Vektor-Matrix-Multiplikation optimierten Vektorprozessor besitzt der NM6403 einen Skalarprozessor, der hauptsächlich für die Berechnung von Adressen benutzt wird, sowie einen DMA-Coprozessor, der zwei sogenannte DSP-Links steuert (Abbildung 3.4). Diese Links sind zum digitalen Signalprozessor TMS320C40 kompatibel und werden für die Interprozessorkommunikation zwischen mehreren Neuro-Chips oder DSPs eingesetzt. Zwei getrennt voneinander arbeitende Memory-Controller mit je 15-Bit-Adreßbus, erlauben den gleichzeitigen Zugriff auf zwei externe Speicher, wobei sowohl DRAMs als auch SRAMs unterstützt werden. Speicherzugriffe erfolgen ausschließlich im 64-Bit-Format.

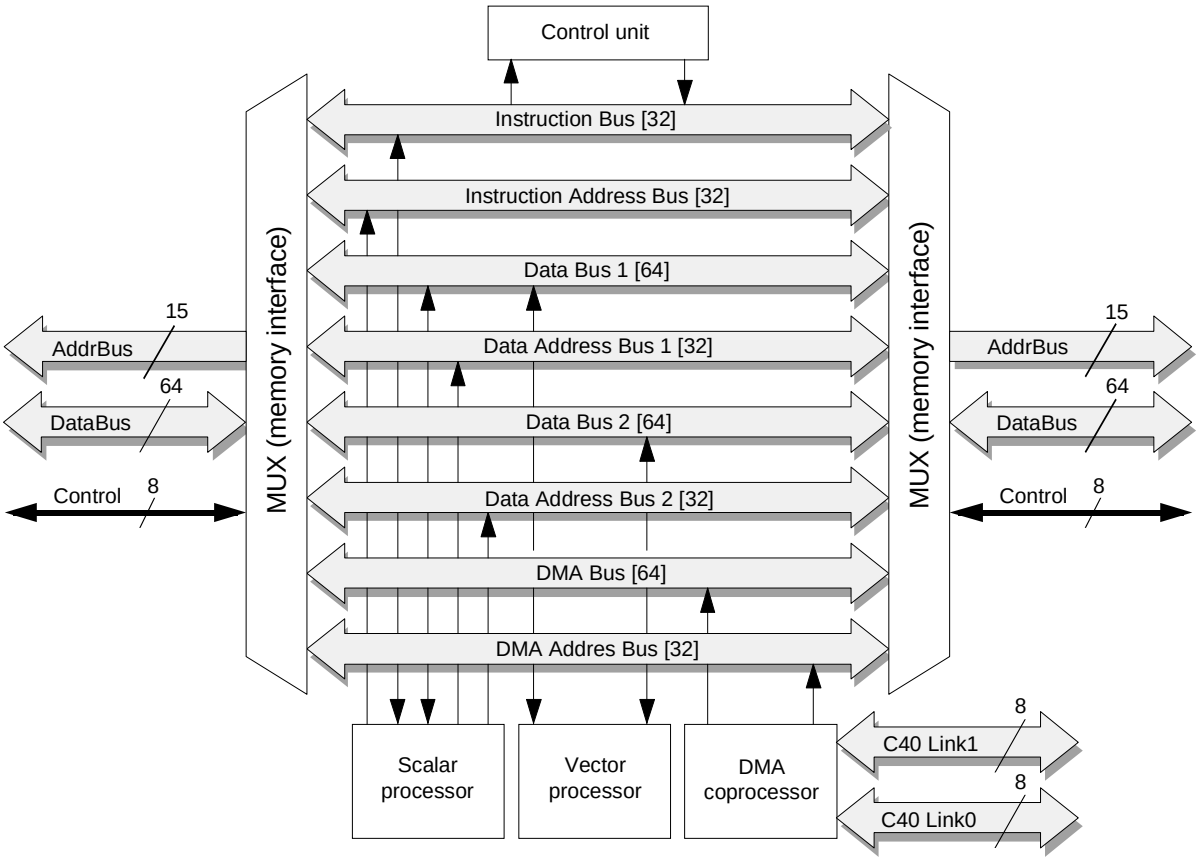


Abbildung 3.4: Architektur des NM6403 nach [Vixne96]

Interessant ist der sogenannte superskalare Aufbau des Vektorprozessors, der je nach gewählter Genauigkeit der Operanden eine mehr oder minder starke Parallelisierung der Operationen erlaubt. Ein 64-Bit-Datenwort am Eingang des Vektorprozessors kann mit insge-

samt 192 im Vektorprozessor gespeicherten Koeffizienten-Bit multiplikativ verknüpft werden. Eine nachfolgende Akkumulation erlaubt die Bildung von Produktsummen, deren Gesamtwortbreite wieder 64 Bit beträgt. Das Attribut superskalar bezieht sich nun darauf, daß die vorhandenen Ressourcen, ein 192 x 64-Bit-Multiplizier-Array, auf vielfältige Art und Weise konfiguriert werden kann. Somit sind einerseits Konfigurationen mit großer Genauigkeit aber geringer Parallelisierung und andererseits solche mit geringer Genauigkeit aber einem hohen Parallelisierungsgrad möglich. Die zugrunde liegende Idee beruht auf der Tatsache, daß die Multiplikation zweier Binärzahlen in Hardware als wiederholte Addition, einschließlich einer Shift-Operation für jeden Summanden, ausgeführt wird.

Bei einer Taktfrequenz von 30 MHz beträgt die maximale Leistung des NM6403 720 MCPS. In dieser Leistungsangabe wurde weder das Laden der Gewichte noch das Laden und Speichern der Aktivitäten berücksichtigt. Ferner bezieht sich die Angabe auf 8-Bit-Operanden. Werden alle diese Fakten beachtet, dann ergibt sich eine Leistung von 60 MCPS, wenn die Operanden gemäß Gleichung (2.21) normiert werden.

Die superskalare Architektur, wie sie beim NM6403 implementiert wurde, hat den entscheidenden Nachteil, daß es keine Überlaufkontrolle gibt. Erzeugt die Multiplikation eines Gewichtes mit einer Aktivität einen unzulässigen Überlauf in das nächste Operandenfeld, werden hierdurch im ungünstigsten Fall alle anderen Ergebnisse des gleichen Berechnungsschrittes verfälscht. Aus diesem Grund kann der NM6403 weit weniger flexibel konfiguriert werden als man zunächst annehmen könnte. In [Vixne96] wird die Verwendung eines angepaßten Lernverfahrens empfohlen, im Rahmen dessen die Größe der Gewichte überwacht wird, so daß Überläufe ausgeschlossen werden können. Hierdurch wird jedoch die Handhabung des Chips erschwert, da nur entsprechend trainierte Netze durch den NM6403 beschleunigt werden können. Ein zweiter, erheblicher Nachteil dieser superskalaren Architektur ist die für einige Netzkonfigurationen unbefriedigende Effizienz. Wird das in [Vixne96] angegebene Netz mit 16-Bit-Operanden auf dem Chip berechnet, so können nur ein Drittel der Ressourcen auf dem Chip genutzt werden.

IBM Neuro-Chip ZISC 036

Zur Beschleunigung von Netzen mit RBF-ähnlichen Neuronen und K-Nearest-Neighbour-Funktionen (KNN) wurde der ZISC036-Prozessor von IBM Frankreich entwickelt [IBM98a, IBM98b]. Typische Einsatzgebiete dieses Neuro-Chips sind die Bilderkennung und -kompression [Trémiolles97] sowie die Signalerkennung bei Radar- und Ultraschallanwendungen.

Der ZISC036 besitzt eine SIMD-Architektur mit 36 parallelen Prozeßelementen, welche 36 Neuronen entsprechen, d.h. der ZISC036 parallelisiert Netze auf der Ebene einzelner Neuronen (Abbildung 3.5). Jede dieser Verarbeitungseinheiten ist in der Lage, den Abstand zwischen Eingangsdaten $\underline{v}$ und lokal gespeicherten Prototypen $\underline{p}$ auf zwei Arten zu bestimmen (3.2), (3.3).

NORM L1

$$distL1 = \sum_{i=1}^{64} abs\left(v_i - p_i\right) \qquad (3.2)$$

NORM LSUP

$$distLSup = \max\left(abs\left(v_i - p_i\right)\right) \;\; \forall i \in \left[1,64\right] \qquad (3.3)$$

In diesem Zusammenhang spricht IBM von einem **Z**ero **I**nstruction **S**et **C**omputer (ZISC), da der Chip lediglich konfiguriert, jedoch nicht programmiert werden kann. Die Kaskadierung mehrerer Chips zur Realisierung größerer Netzwerke wird durch eine Daisy-Chain-Verbindung ermöglicht, die alle Neuronen, auch über die Grenzen eines Chips hinaus, miteinander verbindet. Der Eingaberaum ist jedoch, wie aus Gleichung (3.2) ersichtlich ist, auf 64 Dimensionen begrenzt. Eingangsdaten und Prototypen werden als 8-Bit-Festkomma-Werte verarbeitet. Bei einer maximalen Taktfrequenz von 20 MHz verarbeitet der ZISC036 64 Eingabedaten in 3,7 µs [IBM98a], was einer Leistung von 632 MCPS bzw. 158 MCPS gemäß Gleichung (2.21) entspricht.

Die Stärke des ZISC036 liegt sicherlich in seinem übersichtlichen Aufbau, der hohen Integrationsdichte (Prototypen, d.h. Gewichte werden auf dem Chip gespeichert), der einfachen Handhabung und seiner relativ hohen Verarbeitungsleistung. Die Möglichkeit, Netze durch einen auf dem Chip integrierten Lernalgorithmus zu trainieren, eröffnet viele Anwendungsmöglichkeiten im Bereich adaptiver Systeme. Der begrenzte Speicher für lediglich 64 Prototypen und die Beschränkung auf RBF-ähnliche Netze engen den Einsatzbereich des ZISC036 deutlich ein. Zu beachten ist, daß bedingt durch die L1-Norm lediglich lineare, jedoch keine radialen Gebietsgrenzen in einem Eingaberaum gebildet werden können.

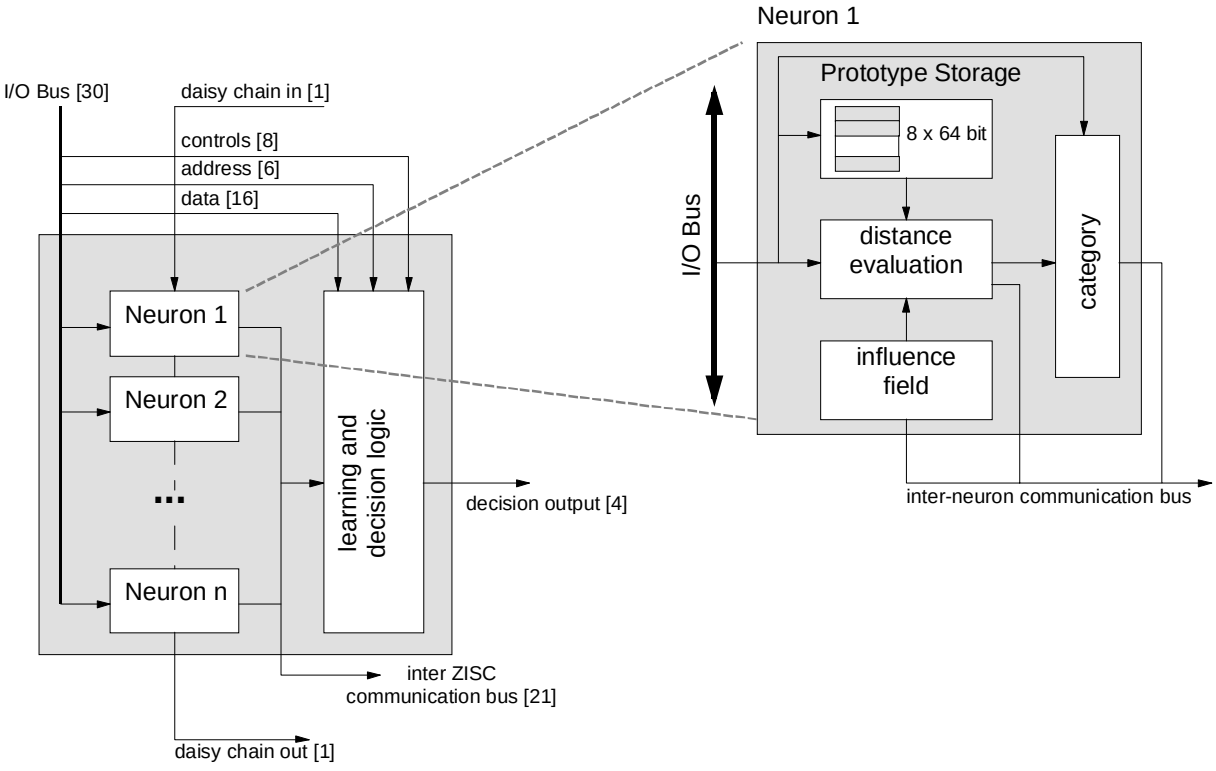


Abbildung 3.5: Architektur des IBM ZISC036 Neuro-Chips

3.2.1 Digitale Multi Model Neuro-Chips

CNAPS 10xx von Adaptive Solutions

Von der Firma Adaptive Solutions, USA wurde der Parallelprozessor CNAPS [Hammers90, Hammers95] entwickelt und heute in verschiedenen Ausführungen mit bis zu 64 einzelnen Prozessorknoten (PN) pro Chip gefertigt. Beim CNAPS-Chip operieren alle parallelen PNs auf unterschiedlichen Daten, führen aber dabei die gleiche Operation aus, so daß CNAPS ein typischer Vertreter der SIMD-Prozessoren ist. Neben einem acht Bit breiten Bus für Eingangsdaten (IN Bus) und einem ebenso breiten Bus für Ausgangsdaten (OUT Bus) besitzt jeder Prozessorknoten einen eigenen, lokalen vier KBytes großen Speicher (SRAM), der Zugriffe mit dem Takt des Prozessors erlaubt. Neben diesen beiden 8-Bit-Bussen, über die die Daten per Broadcasting versendet werden, gibt es zwischen benachbarten PNs lokale Verbindungen, die allerdings nur eine Breite von zwei Bit haben. Diese lokale Interprozessorkommunikation wird hauptsächlich bei Anwendungen aus dem Bereich der Bildverarbeitung, z.B. bei Filteroperationen, genutzt.

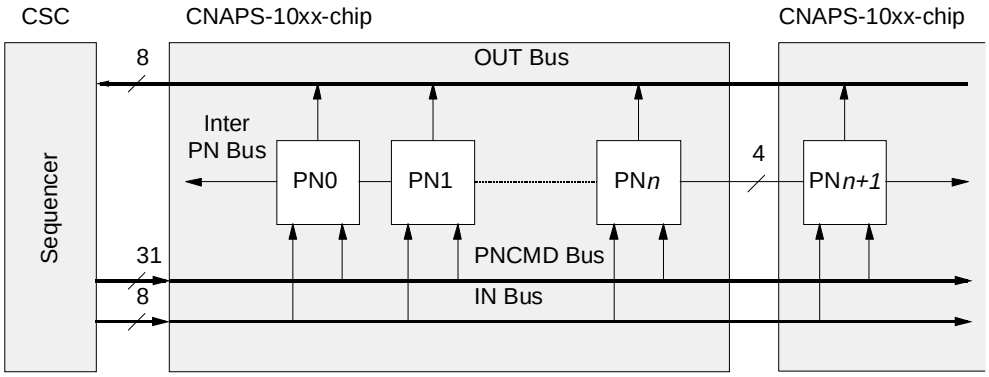


Abbildung 3.6: Architektur des CNAPS-10xx-Parallelprozessors

Trotz der beachtlichen Anzahl von bis zu 64 Prozessoren pro Chip hält sich der Verdrahtungsaufwand in Form von Bussen auf dem Chip in Grenzen, was sich einerseits auf die Verwendung lokaler Speicher und andererseits auf die geringe Busbreite von lediglich acht Bit zurückführen läßt. Ein separates Steuerwerk, ein sogenannter Sequencer-Chip (CSC) [CNAPS95], erlaubt die Steuerung von bis zu acht parallelen Chips mit insgesamt 512 Prozessorknoten. Dieser Sequencer dient außerdem der Kommunikation mit einem Hostsystem und hat einen 32 Bit breiten Datenbus für Zugriffe auf externe Speicher (DRAM).

Die Architektur eines einzelnen Prozessorknotens des CNAPS-10xx-Chips erinnert an einen einfachen DSP (Abbildung 3.7), der vor allem auf die schnelle Ausführung einer Multiply-Add-Sequenz ausgelegt ist. Der Multiplizierer ermöglicht wahlweise eine 8 x 8-Bit-Multiplikation in einem Takt oder eine 8 x 16-Bit-Multiplikation in zwei Takten. Die Verarbeitung von zwei 16-Bit-Operanden ist prinzipiell möglich, sie verursacht aber einen erheblichen Leistungsverlust.

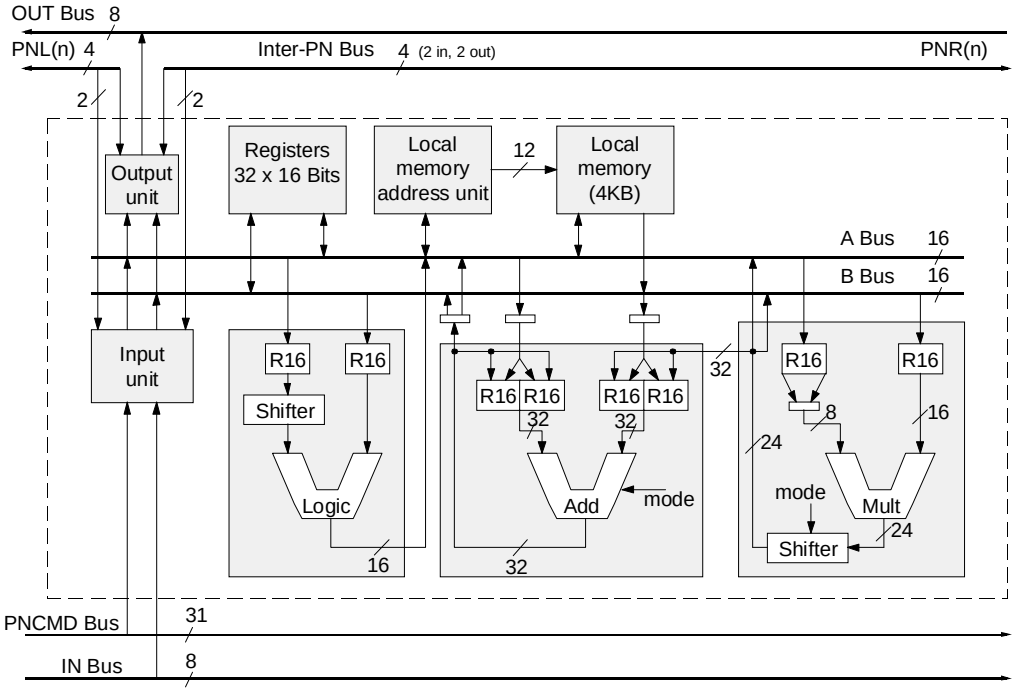


Abbildung 3.7: Architektur eines Prozessor-Knotens beim CNAPS 10xx

Der CNAPS-10xx-Chip ist auf die Beschleunigung vollvernetzter Strukturen ausgelegt, was sich insbesondere durch die Broadcast-Architektur belegen läßt. Damit kann CNAPS eine Schicht eines MLP-Netzes mit dem Aufwand $O(m \cdot n)$ in nahezu m Takten berechnen, falls m die Anzahl der Eingänge, n die Anzahl der Ausgänge und $n \leq p$ (p: Anzahl der parallelen Prozessorknoten) ist. Der an die Architektur eines einfachen Signalprozessors angelehnte Prozessorknoten erlaubt die Berechnung verschiedener Neuronenmodelle nicht zuletzt dadurch, daß sich umfangreichere Algorithmen unter Benutzung der lokalen Register und des lokalen Speichers implementieren lassen.

Durch die direkte Verbindung zwischen dem Ausgang des Multiplizierers und einem Eingang des Addierers entsteht eine Multiply-Add-Pipeline. Damit kann ein CNAPS-Prozessorknoten trotz seines General-Purpose-Charakters pro Takt eine Verbindung eines Multilayer-Perzeptrons berechnen, allerdings nur mit 8-Bit-Daten. Werden statt dessen 16-Bit-Daten (Gewichte und Aktivitäten) verarbeitet, so ändert sich die Situation grundlegend. Die Leistung wird mehr als halbiert, da die Multiply-Add-Pipeline nicht mehr genutzt werden kann. Statt dessen müssen die Operanden über Register und über die internen Busse (A/B) ausgetauscht werden. Ferner ergeben sich Einschränkungen bei der Bestimmung der Ausgangsaktivitäten, da aufgrund des nur vier KBytes großen internen Speichers keine 16-Bit-Look-Up-Tabelle verwendet werden kann.

Zur Programmierung des CNAPS stehen ein C-Compiler, ein Assembler und ein Debugger zur Verfügung [Zell95, Salmen95]. Der C-Compiler wurde um zusätzliche Konstrukte zur Beschreibung paralleler Abläufe erweitert, so daß sich z.B. eine Matrix-Vektor-Multiplikation mit Hilfe einer einzigen Schleife beschreiben läßt [Hammers95]. Untersuchungen an komplexeren Algorithmen [Wahle94] haben aber gezeigt, daß ein C-Programm um den Faktor sieben langsamer läuft als ein vergleichbares, von Hand geschriebenes Assembler-Programm.

MA16 von SIEMENS

Der von SIEMENS entwickelte Array-Prozessor MA16 [Siemens92, Siemens93] ist sowohl von der Flexibilität als auch von der Leistung her mit dem CNAPS-10xx-Chip vergleichbar. Dennoch unterscheiden sich beide Architekturen grundlegend. Während der Chip von Adaptive Solutions, wie bereits erläutert, ein typischer Vertreter der SIMD-Klasse ist, setzt sich die Architektur des MA16 aus mehreren systolischen Komponenten zusammen (Abbildung 3.8).

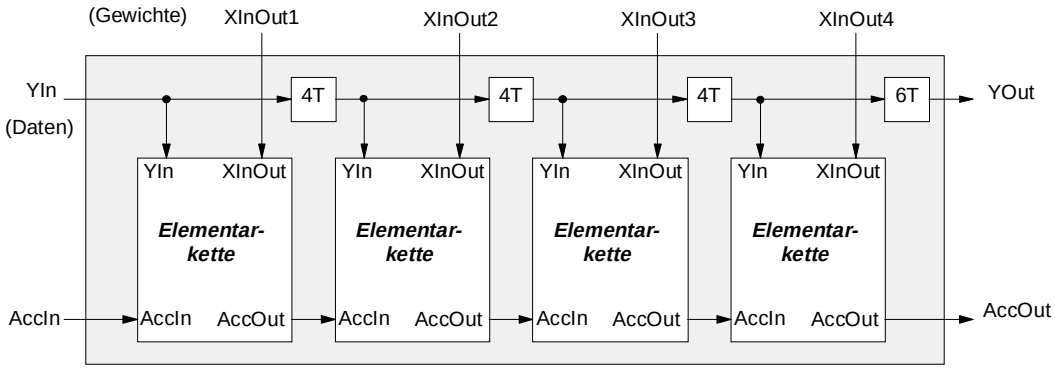


Abbildung 3.8: Struktur des MA16-Chips

Insgesamt sind vier sogenannte Elementarketten auf dem Chip in einer systolischen Kette angeordnet. Jedes dieser vier Basiselemente verarbeitet jedoch keine skalaren Operanden, sondern ausschließlich Operanden in Matrix- und Vektorform. Zentrales Element ist ein Matrix-Matrix-Multiplizierer, der Matrizen der Größe (4 x 4) in einem systolischen Feld miteinander verknüpft. Neben der Multiplikation erlaubt diese Komponente die Addition und die Subtraktion zweier Matrizen. Für die Verknüpfung einer (Sub-) Matrix mit einem Skalar steht ein separater Skalar-Multiplizierer zur Verfügung, der zusätzlich das Quadrieren der Matrixelemente vornehmen sowie deren Absolutbetrag ermitteln kann.

Die Architektur des MA16 erfordert die Zerteilung einer (Gewichts-) Matrix in Submatrizen der konstanten Größe (4 x 4). Dies hat zur Folge, daß im Anschluß an die Verarbeitung im Matrix-Matrix-Multiplizierer die Ergebnis-Submatrizen wieder durch Addition zusammengesetzt werden müssen. Hierfür steht pro Elementarkette ein Matrix-Akkumulator zur Verfügung, der neben intern erzeugten Submatrizen auch extern gespeicherte Matritzen verarbeiten kann (Abbildung 3.9). Jede Elementarkette liest eine (4 x 4)-Gewichts-Submatrix aus einem separaten, externen Speicher (XinOut) und multipliziert diese mit einer (4 x 4)-Aktivitäts-Submatrix, welche elementweise, um jeweils vier Takte verzögert, zwischen benachbarten Elementarketten über den YInOut-Bus übertragen wird. Diese Verzögerung entspricht derjenigen, die ein 48-Bit-Wort zwischen AccIn und AccOut innerhalb einer Elementarkette erfährt. Innerhalb einer Elementarkette werden die (4 x 4)-Submatrizen mit Hilfe einer systolischen Anordnung von Multiplizieren und Addierern in 16 Takten verarbeitet.

Der in einem 1µm-CMOS-Prozeß gefertigte Fullcustom-Chip MA16 erreicht bei 50 MHz Taktfrequenz eine maximale Leistung von 800 MCPS (16-Bit-Operanden). Dies gilt jedoch nur für Operationen in der Art von Gleichung (2.1), bei der Berechnung der L1- und L2-Norm beträgt die Leistung lediglich 200 MCPS. Die Vorteile des MA16 liegen in der externen Speicherung der Gewichte und der leichten Kaskadierbarkeit. In [Ramach91] wird ein

Neuro-Computer auf der Basis von insgesamt 256 MA16-Chips vorgeschlagen, dessen Maximalleistung 128 GCPS (**G**iga **C**onnections **p**er **S**econd) erreichen soll. Der kommerziell verfügbare SYNAPSE-1-Neuro-Computer erreicht immerhin eine Maximalleistung von 5,1 GCPS. Trotz dieser enormen Leistungsfähigkeit haben die auf dem MA16 basierenden Systeme (SYNAPSE-1 und SYNAPSE-2 [Siemens94]) keine hohe Akzeptanz gefunden, was auf die äußert komplexe Handhabung des Chips zurückgeführt werden kann [Ienne95a]. Die Kombination von dreierlei Rechenwerken (Matrix-Multiplizierer, Skalar-Multiplizierer und Akkumulationseinheit) in Verbindung mit der systolischen Verarbeitung erfordern vom Anwender fundierte Kenntnisse über die MA16-Hardware. Im Gegensatz zur SIMD-Architektur des CNAPS 10xx ist beim MA16 durch die systolische Verarbeitungsweise der Einsatz eines um parallele Sprachelemente erweiterten C-Compilers kaum möglich. Ein zweiter, wesentlicher Nachteil ist der Aufwand an externen Bauelementen der nötig ist, um ein Stand-Alone-System mit einem MA16 aufzubauen.

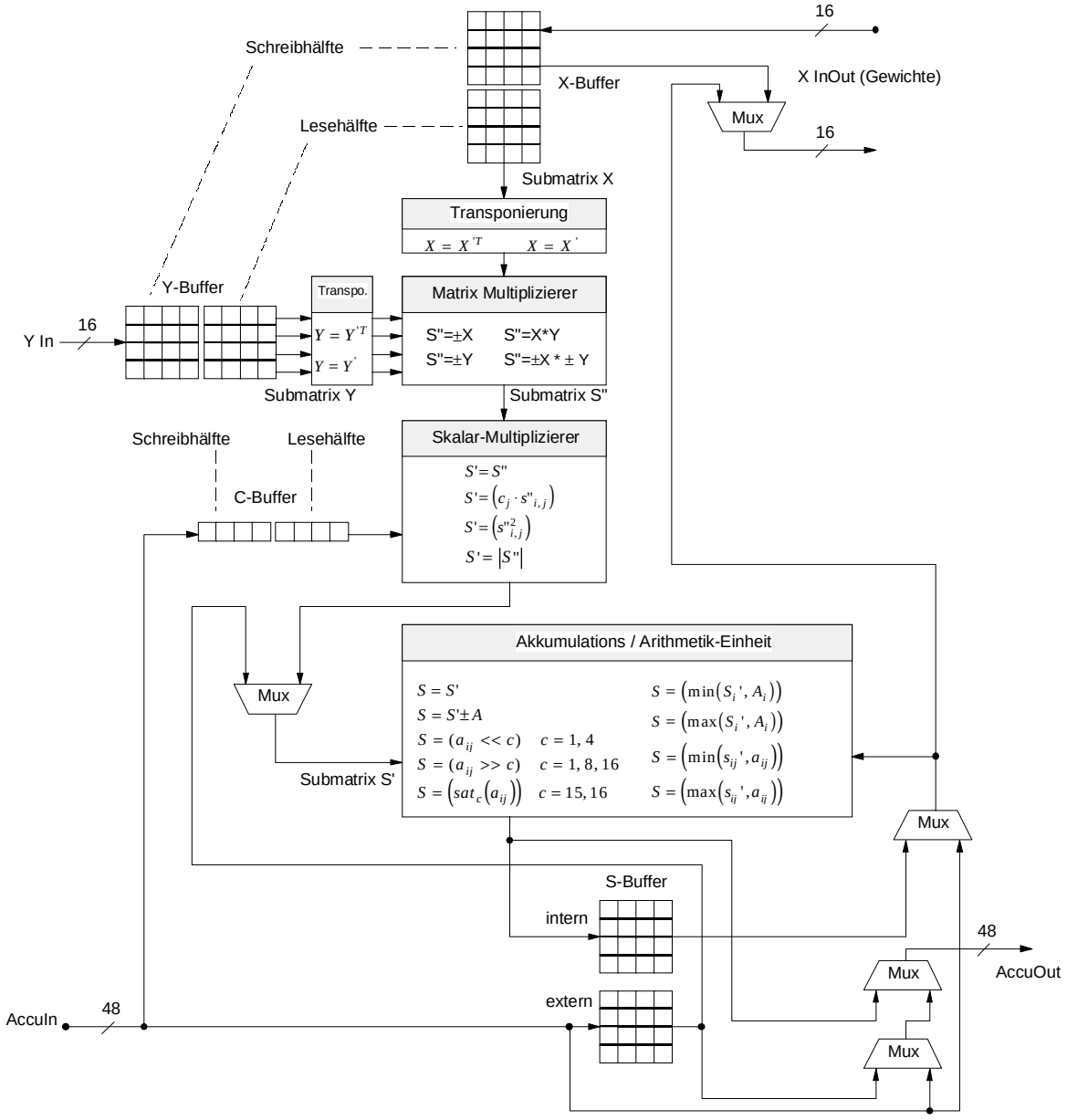


Abbildung 3.9: Architektur einer Elementarkette des MA16

4 Eine neue systolische Busarchitektur

4.1 Einleitung

In Kapitel 2.2 dieser Arbeit wurde eine Klassifikation neuronaler Netze durchgeführt, und es wurde anhand einer Studie erläutert, welche konnektionistischen Modelle für einen Großteil der Anwendungen mit neuronalen Netzen Relevanz besitzen. Die Betrachtung der Hardware-Implementierungen hat gezeigt, daß keine davon sämtlich der in der Einleitung erläuterten Forderungen (hohe Leistung **und** einfache Handhabung **und** niedriger Preis) erfüllt.

In den folgenden Abschnitten wird eine neue Architektur zur Beschleunigung neuronaler Operationen und von Faltungsoperationen der Bildverarbeitung vorgestellt, die in einer kostengünstigen Sea-Of-Gates-Technologie implementiert werden kann, welche aus Anwendersicht die wichtigsten konnektionistischen Modelle unterstützt und eine einfache Handhabung besitzt. Die Benutzerfreundlichkeit zeigt sich sowohl bei der Programmierung, als auch im Aufwand externer Bauelemente. Ausgangspunkt der weiteren Betrachtungen ist eine Analyse parallelisierbarer Algorithmen.

4.2 Analyse parallelisierbarer Algorithmen

4.2.1 Neuronale Algorithmen

Multilayer-Perzeptron (Recall-Phase)

Ausgehend von (2.1) kann ein einzelnes Neuron auf der Ebene einzelner Verbindungen unter Verwendung des verallgemeinerten Neurons aus Kapitel 2.2 mit Hilfe der drei Funktionen h, g und f beschrieben werden (4.1). Es ergibt sich für ein Neuron j in einer Schicht ν eine geschlossene Darstellung (4.2). Faßt man alle Neuronen zwischen zwei aufeinanderfolgenden Schichten ν *-1* und ν eines Multilayer-Perzeptrons (MLP) zusammen (Abbildung 4.1), so kann die Funktion einer Schicht ν eines MLP mit l - *1* verdeckten Schichten in vektorieller Form angegeben werden. Für alle l Schichten des Netzes ergeben sich l Ausdrücke in vektorieller Schreibweise (4.3), welche zusammengefaßt die Funktion des gesamtem MLP beschreiben (4.4). Die verschiedenen Darstellungsformen entsprechen, falls die jeweiligen Ausdrücke sequentiell berechnet werden, den vier Parallelitätsebenen des Kapitels 2.3.

$$\begin{aligned} h &= w_{ji} \cdot o_i \\ g &= \sum\nolimits_i h\left(w_{ji}, o_i\right) = a_j \\ f &= f_{sigmoid}\left(a_j\right) \end{aligned} \qquad (4.1)$$

$$o_j^{[\nu]} = f_{sigmoid}\left(\sum_{i=1}^{n_\nu} w_{ji}^{[\nu]} \cdot o_i^{[\nu-1]}\right) \qquad (4.2)$$

$$\underline{o}^{[1]} = f_{sigmoid}\left(\underline{W}^{[1]} \cdot \underline{o}^{[o]} - \underline{\Theta}^{[1]}\right)$$
$$\vdots$$
$$\underline{o}^{[v]} = f_{sigmoid}\left(\underline{a}^{[v]} - \underline{\Theta}^{[v]}\right) \quad wobei\ gilt \quad \underline{a}^{[v]} = \underline{W}^{[v]} \cdot \underline{o}^{[v-1]} \qquad (4.3)$$
$$\vdots$$
$$\underline{o}^{[l]} = f_{sigmoid}\left(\underline{W}^{[l]} \cdot \underline{o}^{[l-1]} - \underline{\Theta}^{[l]}\right)$$

$$\underline{o}^{[l]} = f_{sigmoid}\left(\underline{W}^{[l]} \cdot f_{sigmoid}\left(\underline{W}^{[l-1]} \cdot \ldots \cdot f_{sigmoid}\left(\underline{W}^{[1]} \cdot \underline{o}^{[0]} + \underline{\Theta}^{[1]}\right) + \ldots\right) + \underline{\Theta}^{[l]}\right) \qquad (4.4)$$

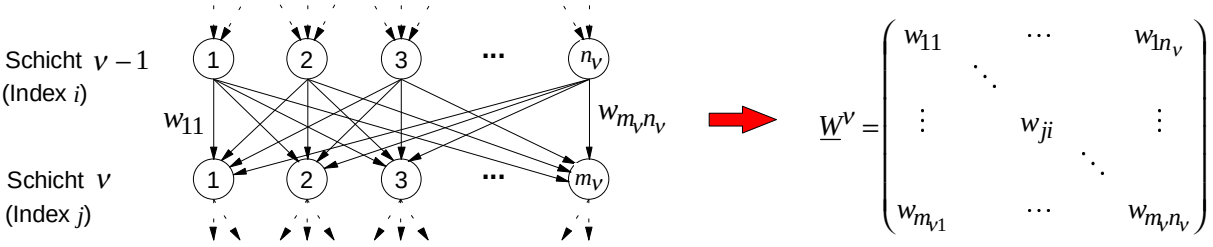


Abbildung 4.1: Zusammenfassen der Gewichte in einer Matrix

Multilayer-Perzeptron (Training)

Von den mittlerweile zahlreichen Trainingsverfahren für Multilayer-Perzeptronen wird im folgenden stellvertretend das Backpropagation-Lernverfahren betrachtet. Es ist mit Sicherheit das am häufigsten verwendete Trainingsverfahren für Multilayer-Perzeptronen. Die für eine Hardware-Implementierung noch interessanten stochastischen Verfahren, deren Implementierungsaufwand um ca. 40% geringer ist, zeigen vor allem bei größeren Netzen ein deutlich langsameres Konvergenzverhalten, das beim heutigen Stand der Technik auch nicht durch entsprechend hohe Taktraten kompensiert werden kann. In [Roth96] wurde gezeigt, daß zwischen den Trainingszeiten bei Backpropagation und einem stochastischen Lernverfahren folgender Zusammenhang gilt:

$$t_{learn}^{stochastisch} \approx \sqrt{\#Gewichte} \cdot t_{learn}^{BackProp} \qquad (4.5)$$

In Kapitel 2.2 wurde bereits in groben Zügen das Backpropagation-Lernverfahren erläutert, dessen sogenannter Vorwärtsschritt der oben dargestellten Recall-Phase entspricht. Die Berechnung der Fehler von Neuronen der verdeckten Schichten (Rückwärtsschritt) kann ähnlich dem obigen Vorgehen auf verschiedenen Parallelitätsebenen dargestellt werden. Der Fehler eines einzelnen Neurons j in einer verdeckten Schicht ν (2.4) kann wie der Vorwärtsschritt mit Hilfe der drei Funktionen des verallgemeinerten Neurons dargestellt werden (4.7). Die Umkehrung des Informationsflusses erfordert die Transponierung der Gewichtsmatrix $\underline{W}^{[v]}$, sofern die Fehlerbestimmung einer gesamten Schicht in vektorieller Schreibweise betrachtet wird (4.7). Auf der Basis der berechneten Fehler kann die Bestimmung der Gewichtsänderungen erfolgen. Werden dem Netz zunächst alle Muster (U) präsentiert, spricht man von Offline- oder Batch-Training (4.9), werden hingegen die Gewichte nach jedem Muster (u) geändert, spricht man von Online-Training (4.8).

$$h = err_i \cdot w_{ij} \qquad (4.6)$$

$$g = \sum_i h\left(err_i, w_{ij}\right)$$

$$f = f'_{sigmoid}\left(a_j\right) \cdot g\left(h\left(erri, w_{ij}\right)\right)$$

$$\underline{err}^{[\nu]} = f'\left(\underline{a}^{[\nu]}\right) \circ \left(\underline{W}^{[\nu+1]^T} \cdot \underline{err}^{[\nu+1]}\right) \qquad \text{(siehe Anmerkung[8])} \qquad (4.7)$$

$$\Delta_e \underline{W}^{[\nu]} = \eta \cdot \underline{err}^{[\nu]} \cdot \underline{o}^{[\nu]^T} \quad e \in [1, U] \qquad (4.8)$$

$$\Delta \underline{W}^{[\nu]} = \eta \cdot \sum_{u=1}^{U} \underline{err}_u^{[\nu]} \cdot \underline{o}^{[\nu]^T} \qquad (4.9)$$

Eine Parallelisierung läßt sich auch beim Backpropagation-Lernverfahren auf verschiedenen Ebenen durchführen, wobei die parallele Bearbeitung eines gesamtes Netzes (Ebene 3) aus den bereits erläuterten Gründen nicht genauer betrachtet wird. Zusätzlicher Aufwand entsteht bei der Berechnung der Gewichtsänderungen, deren mathematische Beschreibung von den beiden bisher betrachteten Algorithmen (Recall und Fehlerberechnung) abweicht.

Kohonen-Netze (Recall-Phase)

Die Berechnung der euklidischen Distanz (2.9) als elementarer Bestandteil dieser Netz-Klasse läßt sich ebenfalls mit Hilfe der drei Funktionen des verallgemeinerten Neurons darstellen (4.10). Eine geschlossene vektorielle Darstellung auf der Ebene der gesamten Kohonen-Schicht existiert nicht, was aber nicht bedeutet, daß es hierfür keine Parallelisierung gäbe. Eine vektorielle Darstellungsform erleichtertet jedoch die Definition einer geeigneten Architektur.

$$h = \left(o_i - w_{ji}\right)^2$$

$$g = \sum_i h\left(o_i, w_{ji}\right) \qquad (4.10)$$

$$f \equiv g$$

Die Berechnung der Quadratwurzel in (2.9) kann entfallen, da lediglich der Vektor mit dem geringsten Abstand (Argument der Minimumsfunktion) und nicht der Abstand selbst gesucht ist. Das Aufsuchen des kleinsten Abstandes (2.10) läßt sich dann Parallelisieren, wenn auf alle berechneten Abstände simultan zugegriffen werden kann. Eine Möglichkeit hierfür bestünde in der Verwendung eines Binärbaumes.

RBF-Netze (Recall-Phase)

Auch die Recall-Phase der RBF-Netze erfordert die Berechnung des euklidischen Abstandes zwischen Eingangsdaten und Prototypen (2.9), wobei als Aktivierungsfunktion eine Gaußfunktion verwendet wird. Die Berechnung der Quadratwurzel läßt sich leicht umgehen, da in (2.12) lediglich das Quadrat des Arguments benötigt wird. Somit ergibt sich in Ergänzung zu (4.10) für die Funktion f die Gleichung (4.11).

[8] „$\circ$“ multipliziert zwei Vektoren elementweise (Hadamard-Multiplikation)

$$f = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-\left|g\left(h\left(o_i, w_{ji}\right)\right)\right|}{2\sigma^2}} \qquad (4.11)$$

Die Funktion der Ausgangsschicht stellt eine gewichtete Summation dar, deren algorithmische Beschreibung auf der Ebene einzelner Verbindungen, Neuronen oder der gesamten Schicht den Gleichungen (2.1), (4.1) und (4.2) entspricht, mit dem Unterschied, daß eine lineare Aktivierung $f \equiv g$ verwendet wird.

4.2.2 Algorithmen der digitalen Bildverarbeitung

In der digitalen Bildverarbeitung werden häufig lineare Filteroperationen eingesetzt, die auf einer diskreten Faltung beruhen [Jaehne97]. Mathematisch gesehen beschreibt eine Faltung die Verknüpfung eines Bildpunktes mit seiner Umgebung. Alle Grauwerte der umliegenden Punkte in einer begrenzten Umgebung werden mit den entsprechenden Koeffizienten einer Filtermaske $\underline{H}$ multipliziert, und anschließend werden die entstehenden Produkte addiert. Die resultierende Summe entspricht dem neuen Grauwert b_2 des betrachteten Pixels b_1 an einer Stelle (n,m) des Bildes.

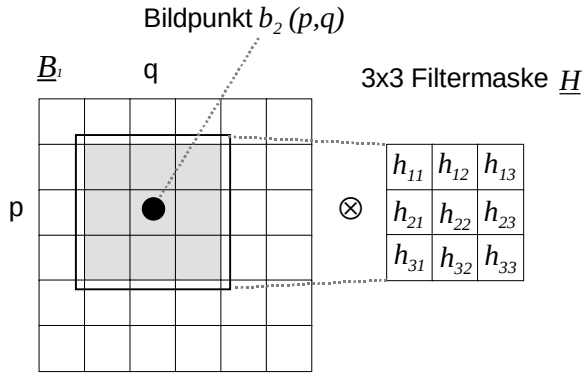


Abbildung 4.2: Beispiel einer zweidimensionalen Faltung unter Verwendung einer 3x3-Maske

Im allgemeinen Fall besitzt die Maske eine ungerade Anzahl von Spalten bzw. Zeilen ($2r+1$). Die zweidimensionale, diskrete Faltung (Abbildung 4.2) ist in (4.12) und (4.13) dargestellt.

$$b_2(p,q) = \underline{B}_1 \otimes \underline{H} \qquad (4.12)$$

$$b_2(p,q) = \sum_{p'=-r}^{r} \sum_{q'=-r}^{r} b_{1_{q+p',p+q'}} \cdot h_{r+p',r+q'} \qquad (4.13)$$

Diese zweidimensionale Betrachtung läßt sich leicht auf eine eindimensionale Adressierung abbilden. Hierzu werden die Zeilen der Filtermaske ebenso wie die Zeilen des Bildbereichs als Vektoren aufgefaßt (Abbildung 4.2), die Indizes p' und q' durch einen Index i und die Position (p,q) durch j ersetzt, so daß sich (4.14) ergibt.

$$b_2(j) = \sum_{i=1}^{(2r+1)^2} b_{1_i} \cdot h_i \qquad (4.14)$$

Aus dieser Darstellung kann nun unmittelbar die Darstellung auf der Basis der drei Funktionen des verallgemeinerten Neurons (Kapitel 2.3) gewonnen werden (4.15).

$$\begin{aligned} h &= b_{1_i} \cdot h_i \\ g &= \sum_i h\left(b_{1_i}, h_i\right) \\ f &\equiv g \end{aligned} \qquad (4.15)$$

4.2.3 Auswertung

Die Vielfältigkeit der im letzten Abschnitt dargestellten Algorithmen läßt noch nicht zu, hieraus unmittelbar eine geeignete Architektur abzuleiten. Deshalb sollen in einem ersten Schritt die Algorithmen zusammengestellt werden, die eine große Ähnlichkeit aufweisen, um daraus dann eine geeignete Architektur abzuleiten. Ähnlichkeit weisen vor allem

- MLP-Recall-Phase
- MLP-Training (Fehler der verdeckten Schichten)
- RBF-Netze (verdeckte Schicht und Ausgangsschicht)
- Kohonen-Netze (Distanzbestimmung)
- Faltungsoperationen

auf. Die Minimumsuche der Kohonen-Netze unterscheidet sich von diesen Algorithmen grundlegend und erfordert deshalb eine gesonderte Behandlung.

Die fünf Algorithmen die eine große Ähnlichkeit aufweisen (s.o.), lassen sich alle mit Hilfe der drei Funktionen h, g und f des verallgemeinerten Neurons beschreiben, wobei alle Algorithmen sogar die gleiche g-Funktion besitzen, so daß sich eine vereinfachte Darstellung (4.16) ergibt.

$$o_j^{[\nu+1]} = f\left(\sum_i h\left(w_{ji}, o_i^{[\nu]}\right)\right) \qquad (4.16)$$

Eine vollständige Parallelisierung des Ausdruckes (4.16) hätte zur Folge, daß jede der möglichen h-Funktionen gleich mehrfach in Hardware implementiert werden müßte. Das bedeutet, daß sämtliche Verbindungen eines Netzes (parallel) in Hardware implementiert werden müßten. Sicherlich ist es denkbar durch Mehrfachnutzung den Overhead zu verkleinern, dennoch multipliziert sich dieser mit der Anzahl paralleler Elemente. Diese nehmen bei der höchsten Form der Parallelisierung mit der Anzahl der Verbindungen in einem neuronalen Netz zu.

Günstiger erscheint es daher, eine Parallelisierung auf Basis der Summen

$$g = \sum_i h\left(w_{ji}, o_i^{[v]} \right) \qquad (4.17)$$

d.h. auf Neuronenebene (Stufe 1) durchzuführen. Die gesamte Verarbeitungszeit eines Neurons bei n Neuronen in der vorherigen Schicht setzt sich aus den beiden Zeiten zur Berechnung der Summe g und der Funktion f zusammen (4.18). Der jeweilige Anteil an der Gesamtzeit ist in der Abbildung 4.3 dargestellt.

$$\begin{aligned} t_\Sigma &\cong n \\ t_f &\cong const = 1 \\ t_{ges} &= t_\Sigma + t_f \end{aligned} \qquad (4.18)$$

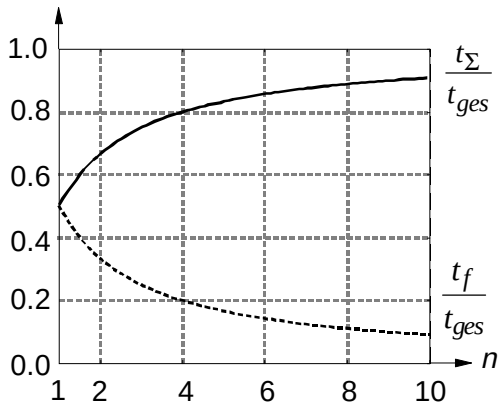


Abbildung 4.3: Anteil der beiden Zeiten an der Gesamtzeit

Für hinreichend große n ($n > 5$) ist eine vollständige Parallelisierung der f-Funktionen daher keinesfalls notwendig. Es genügt völlig, für alle Neuronen einer Schicht eine Implementierung der f-Funktion zur Verfügung zu stellen. Dies scheint auch aus Sicht des Amdahl'schen Gesetzes plausibel zu sein, denn bei m parallelen Neuronen ermöglichen m parallele f-Funktionen bei weitem keine m-fache Beschleunigung. Auch die Algorithmen selbst erfordern höchstens schichtweise unterschiedliche Aktivierungsfunktionen f. Für kleine Werte von n ($n \leq 5$) ist eine spezielle Hardware-Lösung ohnehin nicht erforderlich, da die Leistung eines sequentiellen Prozessors genügen würde.

Die bisherigen Betrachtungen in diesem Abschnitt 4.2.3 ließen die Frage offen, ob eine vollständige Parallelisierung aller Neuronen eines Netzes sinnvoll ist, oder ob gegebenenfalls eine nur schichtweise Parallelisierung vorteilhaft wäre. Eine vollständige Parallelisierung unter Verwendung eines Pipeline-Schemas hat den entscheidenden Vorteil, daß auf der Basis der gewählten Parallelisierungsstufe die kürzeste Verarbeitungszeit erreicht wird. Auf der anderen Seite ergibt sich hierbei das Problem, daß die Abbildung eines (beliebigen) Netzes auf die vollständig parallele Hardware eine zweidimensionale Anpassung erfordert (Anzahl der Neuronen pro Schicht, Anzahl der Schichten pro Netz). Um eine ineffiziente Nutzung der Hardware zu vermeiden, wird im folgenden eine Architektur zur Parallelisierung von Operationen einer einzelnen Schicht vorgestellt.

4.3 Ein neues Kommunikationsschema

4.3.1 Optimierung der Kommunikation

Die Analyse der zu implementierenden Algorithmen hat gezeigt, daß sich die parallelisierbaren Teile in Form von Gleichung (4.17) darstellen lassen. Ausgangspunkt der folgenden Betrachtungen ist eine hieraus resultierende einfache schematische Darstellung der parallelen Architektur zum Datenaustausch in einem Neuro-Chip in Form eines Ablaufdiagramms (Abbildung 4.4).

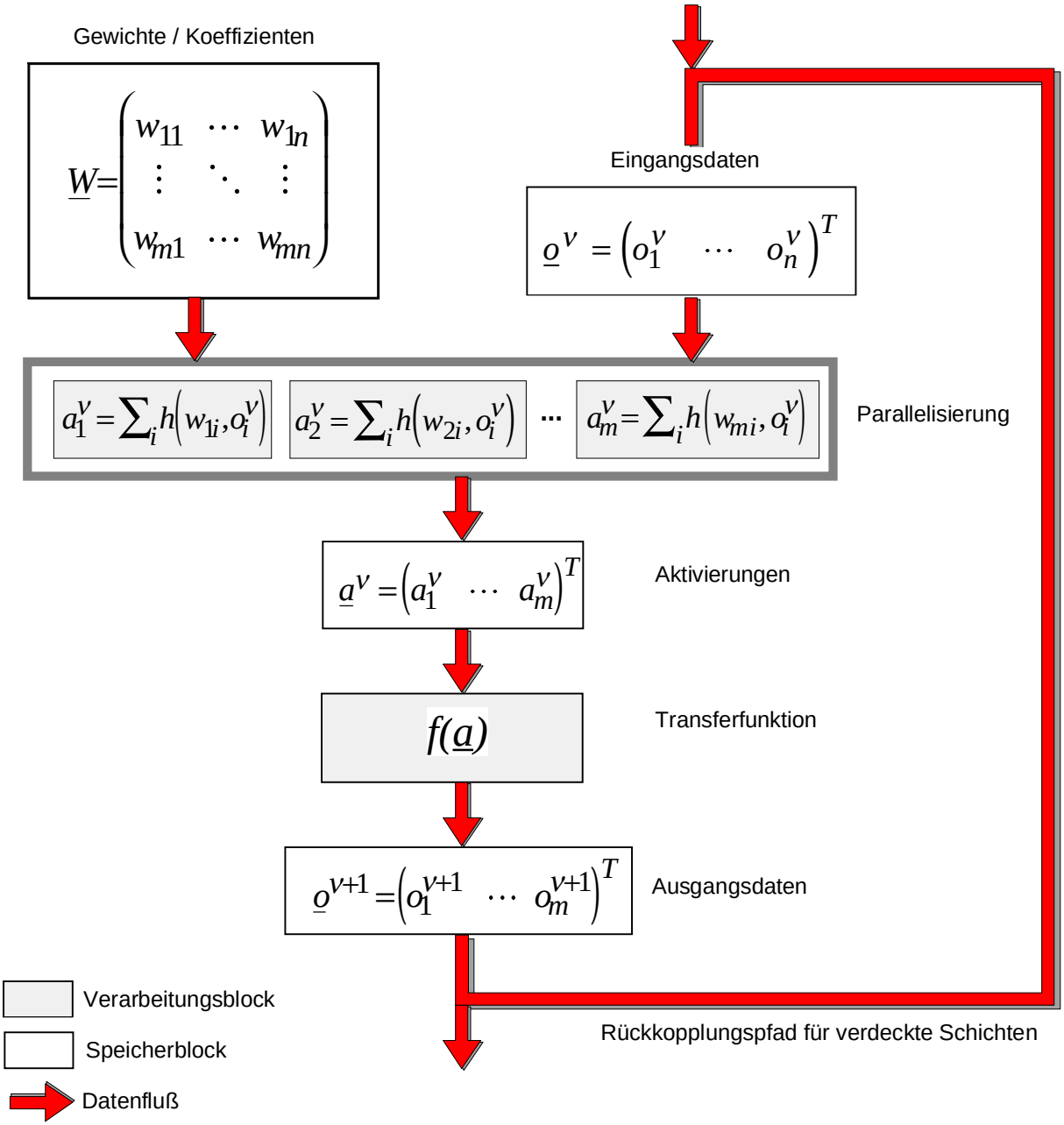


Abbildung 4.4: Schematische Darstellung des Datenflusses der parallelen Architektur

Im Hinblick auf die Implementierung in einem Gate-Array können folgende Randbedingungen identifiziert werden:

Effiziente Nutzung paralleler Ressourcen auf dem Chip

Um ein möglichst gutes Kosten / Nutzenverhältnis zu erzielen, muß durch die Architektur gewährleistet sein, daß möglichst alle parallelen Ressourcen auf dem Chip (gleichzeitig) genutzt werden können, soweit dies im Rahmen des zu implementierenden Algorithmus möglich ist.

Vermeiden von Gewichtsspeichern auf dem Chip

Da für unterschiedliche Eingangsdaten dieselben Gewichte verwendet werden können, wäre es für die Kommunikation optimal, wenn der Gewichtsspeicher auf dem Chip implementiert werden könnte. Die Speicherung von Gewichten auf dem Chip hat aber zwei entscheidende Nachteile:

- Die Größe der Operanden wird durch den Speicher auf dem Chip begrenzt und kann nicht erweitert werden.

- Die Implementierung großer RAM-Blöcke auf dem Chip ist im allgemeinen nur bei Full-Custom-Entwürfen (wie z.B. dem CNAPS 10xx) möglich. Beim heutigen Stand der Technik ist Speicher auf dem Chip (embedded RAM) wesentlich teurer als externer Speicher.

Um ein Gefühl für die erforderliche Speichergröße zu geben, soll folgendes Beispiel betrachtet werden: Ein MLP1-Netz mit z. B. 512 Eingangsneuronen, 128 verdeckten Neuronen und 512 Ausgangsneuronen benötigt bei 16 Bit Auflösung einen Speicherplatz von 256 KBytes für die Gewichte. Weiterer Speicherbedarf ergibt sich, falls die Funktion f in Form einer Tabelle auf dem Chip implementiert werden soll. Im Falle von 16 Bit Adressen bzw. Daten erfordert dies weitere 128 KBytes. Es wird deutlich, daß der Speicherplatzbedarf die derzeitigen (kostengünstigen) technischen Möglichkeiten bei weitem übersteigt. Da sich die Wortbreite der Gewichte nicht auf 8 Bit, 4 Bit oder sogar nur 1 Bit verringern läßt, muß somit auf RAM-Blöcke auf dem Chip verzichtet werden.

Minimierung der Anzahl externer Speicher und I/O Busse

Jedes externe Bauteil erhöht die Gesamtkosten und erschwert die Handhabung des Systems auf der Hardwareebene (Beispiel: Siemens MA16 bzw. Synapse1/2). Einen weiteren Kostenfaktor stellt neben der Größe des DIEs die Größe des Gehäuses, d.h. die Anzahl der Pins dar. Ziel ist es daher, die Anzahl externer Bauelemente und die Anzahl der Pins eines Neuro-Chips so weit wie möglich zu reduzieren.

Leichte Skalierbarkeit der Architektur

Zu einer einfachen Handhabung gehört auch die leichte Skalierbarkeit einer Architektur, die sich in einem möglichst geringen Aufwand in Hard- und Software widerspiegelt.

4.3.2 Einsatzmöglichkeiten bekannter Architekturen

Systolische Architekturen in ihrer ursprünglichen Form, deren Einsatz für die Parallelisierung des in Abbildung 4.4 dargestellten Schemas denkbar wäre (Kette, Ring, - Kapitel 2.4.4), sind nicht in der Lage, alle diese Forderungen gleichzeitig zu erfüllen und dabei unterschiedlich komplexe h-Funktionen (gewichtete Summe, euklidischer Abstand) zu parallelisieren (4.1), (4.15).

Ein systolischer Ring läßt zwar eine einfache Partitionierung durch seine zyklische Verarbeitung zu, doch müssen die Koeffizienten entweder in lokalen Speichern auf dem Chip oder in externen, verteilten Speichern außerhalb des Arrays verwaltet werden, wofür zahlreiche I/O-Busse erforderlich wären. Ein zweiter erheblicher Nachteil ist die unmittelbare Abhängigkeit der Verarbeitungszeit von der Kommunikationszeit. Um die volle Parallelität eines systolischen Ringes zu nutzen, muß die Übertragung einer Partialsumme genauso lange dauern wie die Verknüpfung eines Gewichtes mit einem Aktivitätswert in einem Prozeßelement des Arrays (Abbildung 4.5). Dies scheint vor dem Hintergrund unterschiedlich komplexer h-Funktionen eine nur schwer zu erfüllende Forderung zu sein, da z.B. die Multiplikation eines Gewichts mit einem Aktivitätswert und anschließender Akkumulation eventuell in einem Takt ausgeführt werden kann, während die Berechnung eines euklidischen Abstandes (Subtraktion, Quadrieren, Akkumulation) einen wesentlich größeren Rechenaufwand darstellt.

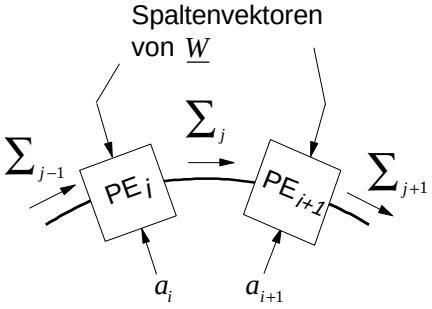


Abbildung 4.5: Ausschnitt aus einem systolischen Ring

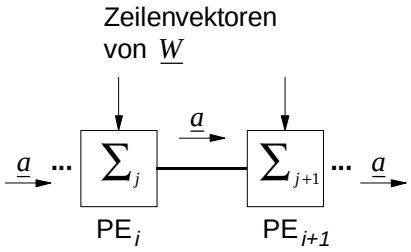


Abbildung 4.6: Ausschnitt aus einer systolischen Kette

Bei der systolischen Kette kann eine Entkopplung von Kommunikations- und Verarbeitungszeit erfolgen, wenn jedes Prozeßelement eine Partialsumme berechnet (Abbildung 4.6). Die systolische Verkettung der Prozeßelemente hat aus technologischer Sicht den Vorteil, daß keine globalen Busse erforderlich sind. Damit lassen sich kapazitive Lasten vermeiden, die sich durch Parallelschaltung der Prozeßelemente ergeben. Dennoch hat auch die systolische Kette das Problem, daß Gewichte entweder lokal auf dem Chip gespeichert werden oder aber von externen, verteilten Speichern bezogen werden müssen.

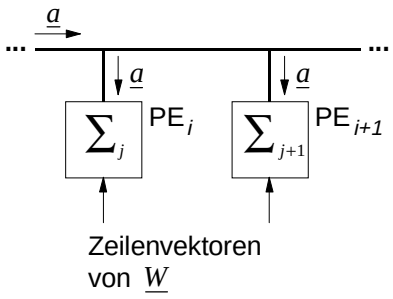


Abbildung 4.7: Ausschnitt aus einer Broadcasting-Struktur

Auch eine feste Verbindungstopologie ohne Pipelining ist nicht in der Lage, alle genannten Forderungen zu erfüllen. Von allen möglichen Topologien ist mit Sicherheit eine eindimensionale Anordnung von Prozeßelementen mit einem One-To-All-Broadcast-Bus am geeignetsten (Abbildung 4.7). Diese Architektur erlaubt zwar eine äußert leichte Skalierung, doch hat man auch hier das Problem, Gewichte entweder intern, oder extern in verteilten Speichern zu verwalten.

4.3.3 Das neue SAND-Kommunikationsschema

Das zentrale Problem bei allen drei betrachteten Architekturen besteht im unmittelbaren Zusammenhang zwischen der Anzahl der Prozeßelemente und der hieraus resultierenden Anzahl verteilter Speicher, welche entweder auf dem Chip oder extern implementiert werden müssen. Dieses Problem wurde in ähnlicher Weise schon beim MA16 gelöst, der jedoch ausschließlich für die Matrix-Matrix-Multiplikation eine geschickte Parallelisierung auf der Basis eines systolischen Arrays besitzt [Ramach92], nicht jedoch für die Berechnung des euklidischen Abstandes. Bei der Betrachtung in dieser Arbeit kommt aber hinzu, daß nicht nur die Multiplikation (4.1), sondern allgemein Funktionen in der Art von Gleichung (4.17) parallelisiert werden sollen.

Anstelle von k Speichern bei k parallelen Prozeßelementen erfordert die neue, im folgenden erläuterte Architektur lediglich einen gemeinsamen Speicher für die Gewichte, der über einen einzigen Bus die parallelen Prozeßelemente mit Koeffizienten versorgt, ohne hierdurch den Datendurchsatz im Vergleich zur verteilten Speicherung zu verringern. Kernstück der neuen Architektur ist ein hybrides Kommunikationsschema, das sowohl eine systolische Kette als auch einen Broadcast-Bus verwendet (Abbildung 4.8).

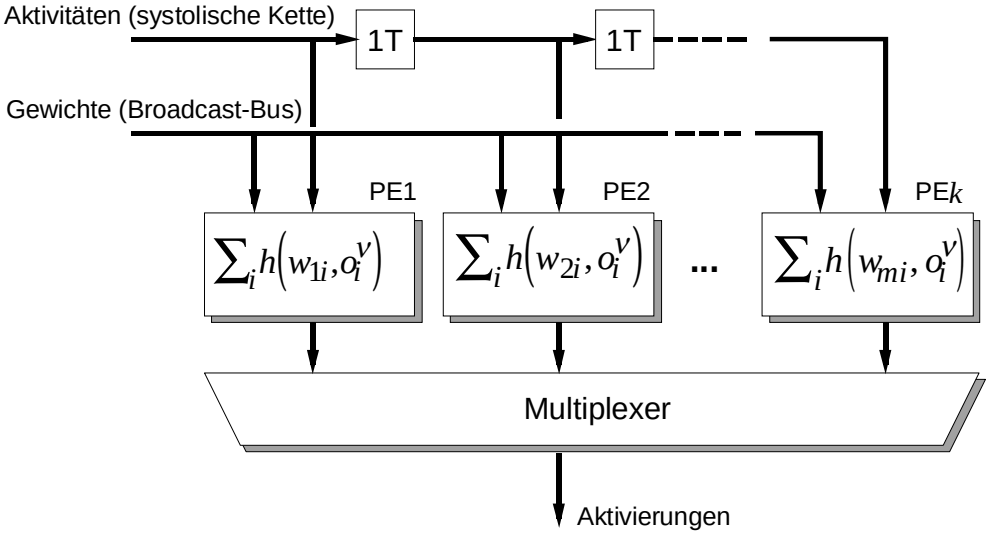


Abbildung 4.8: Schema der neuen Architektur

Um die parallelen Gewichtsbusse, die bei den bekannten Architekturen erforderlich waren, zu einem Bus zusammenfassen zu können, ist eine überlappende Verarbeitung mehrerer Eingangsdaten notwendig [Fischer96b]. Aus der Sichtweise neuronaler Netze bedeutet dies, daß eine synaptische Verbindung gleich für mehrere Aktivitäten berechnet werden muß. Formal gesehen werden hierfür mehrere Muster (k), jeweils repräsentiert durch einen Spalten-Vektor $\underline{o}^{[v]} = (o_1^{[v]} \ldots o_n^{[v]})^T$, zu einer Matrix

$$\underline{O}^{[v]} = \left(\underline{o}_1^{[v]} \quad \cdots \quad \underline{o}_k^{[v]}\right) = \begin{pmatrix} o_{11}^{[v]} & \cdots & o_{1k}^{[v]} \\ \vdots & \ddots & \vdots \\ o_{n1}^{[v]} & \cdots & o_{nk}^{[v]} \end{pmatrix} \qquad (4.19)$$

zusammengefaßt. Über den Broadcast-Bus werden die Gewichte der Matrix $\underline{W}$ (m x n) spaltenweise[9] eingelesen. Dabei erfolgt keine One-To-All-Kommunikation, sondern eine One-To-One-Kommunikation zwischen dem Gewichtsspeicher und einem der Prozeßelemente. Beginnend mit dem ersten Prozeßelement, erfolgt die Auswahl eines Empfängers in zyklischer Reihenfolge, so daß bei k parallelen Elementen nach $k+1$ Takten wieder ein Gewicht an das erste Prozeßelement übertragen wird. Während dieser k Takte werden die Aktivitäten aus der $\underline{O}$-Matrix zeilenweise ausgelesen, d.h. es wird genau eine Zeile der Aktivitätsmatrix verarbeitet. Diese Zeile entspricht den k aufeinanderfolgenden Aktivitäten des ersten Neurons der sogenannten Sendeschicht. Die k-1 Takte, die in jedem Prozeßelement vergehen, bis ein neues Gewicht eingespeichert wird, werden also dazu genutzt, $k-1$ weitere Aktivitäten ein und desselben Sendeneurons zu verarbeiten. Die systolische Verkettung der Prozeßelemente (PE) ist notwendig, um die Elemente der Matrix $\underline{O}$ jeweils um einen Takt verzögert an das Nachbar-PE weiterzuleiten. Das folgende Schema zeigt die Berechnungsschritte der einzelnen Prozeßelemente für zunächst beliebige h-Funktionen.

	PE1	PE2	...	PEk
Takt 1	$\Sigma_1 += h(w_{11}, o_{11})$	-	...	-
Takt 2	$\Sigma_2 += h(w_{11}, o_{12})$	$\Sigma_1 += h(w_{21}, o_{11})$	...	-
...	...	...	...	-
Takt k	$\Sigma_k += h(w_{11}, o_{1k})$	$\Sigma_{k-1} += h(w_{21}, o_{1k-1})$	...	$\Sigma_1 += h(w_{k1}, o_{11})$
Takt k+1	$\Sigma_1 += h(w_{12}, o_{21})$	$\Sigma_k += h(w_{21}, o_{1k})$	...	$\Sigma_2 += h(w_{k1}, o_{12})$
...	...	...	...	...
Takt 2k	$\Sigma_k += h(w_{12}, o_{2k})$	$\Sigma_{k-1} += h(w_{12}, o_{2k-1})$	...	$\Sigma_1 += h(w_{k2}, o_{21})$
Takt 2k+1	$\Sigma_1 += h(w_{13}, o_{31})$	$\Sigma_k += h(w_{22}, o_{2k})$	...	$\Sigma_2 += h(w_{k2}.o_{22})$
...	...	...	...	...
Takt 3k	$\Sigma_k += h(w_{13}, o_{3k})$	$\Sigma_{k-1} += h(w_{22}, o_{3k-1})$		$\Sigma_1 += h(w_{k3}, o_{31})$

Tabelle 4.1: Berechnungsschritte der Prozeßelemente

Am Beispiel von vier parallelen Prozeßelementen ist das resultierende Zeitdiagramm für die beiden Busse (Aktivitäten und Gewichte) in Abbildung 4.9 dargestellt.

[9] spaltenweise bedeutet: die Gewichte von einem Neuron zu allen Neuronen der nächsten Schicht.

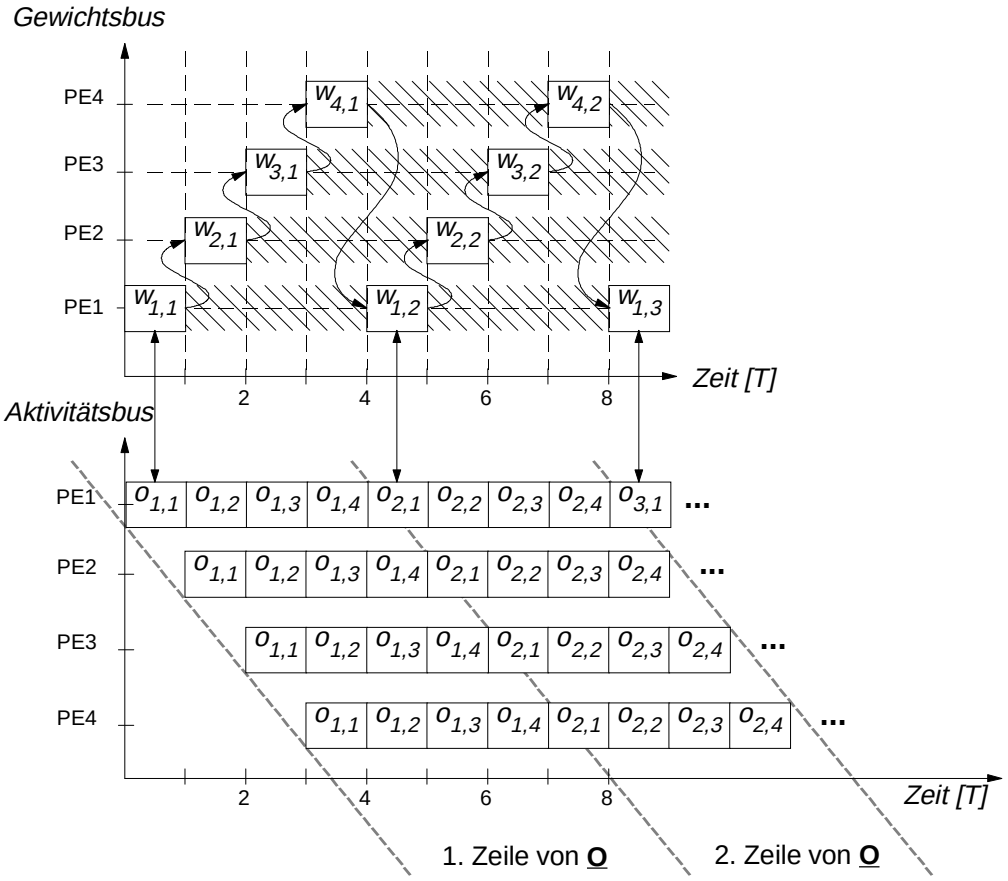


Abbildung 4.9: Zeitdiagramm für vier parallele Prozeßelemente. Oben: Gewichtsbus wird im Zeitmultiplexverfahren genutzt. Unten: Aktivitätsbus wird parallel genutzt.

Bedingt durch das neue Kommunikationsschema, muß die Anzahl der simultan zu verarbeitenden Muster der Anzahl der parallelen Prozeßelemente (k) angepaßt sein. Aus didaktischen Gründen soll zunächst vorausgesetzt werden, daß die Anzahl der Neuronen (m) in der zu berechnenden Schicht ν der Anzahl paralleler Prozeßelemente (k) entsprechen soll. Diese Einschränkung wird später bei der Betrachtung des Partitionierungsproblems aufgehoben.

4.3.4 Vorteile gegenüber bekannten Architekturen

Betrachtet man die Multiplikation (4.1) auf der Ebene einer gesamten Schicht eines Netzes (4.3), so wird deutlich, daß die Matrix-Vektor-Multiplikation durch eine Matrix-Matrix-Multiplikation ersetzt wurde. Es stellt sich die berechtigte Frage, weshalb nicht von vornherein ein planares systolisches Feld (Kapitel 2.4.4) verwendet wurde, das bekanntermaßen für die Beschleunigung der Matrix-Matrix-Multiplikation geeignet ist. Dagegen können drei Argumente angeführt werden.

Erstens hat ein planares systolisches Feld der Größe (n x n) $2n$ Busse (n Eingänge und n Ausgänge), und zweitens gestaltet sich die Skalierung einer solchen zweidimensionalen Architektur weit schwieriger, als dies bei einer eindimensionalen Architektur der Fall ist, da die Größe der Operanden durch die Größe des planaren systolischen Feldes vorgegeben sind.

Der entscheidende Vorteil der neuen Architektur gegenüber einem systolischen Feld liegt darin, daß beim planaren systolischen Feld Zwischenergebnisse von einem PE zum nächsten PE weitergereicht werden, während bei der neuen Architektur Zwischenergebnisse in den PEs verbleiben und nicht zum Nachbar-PE weitergereicht werden müssen. Dadurch gelingt es, Kommunikations- und Berechnungszeiten zu entkoppeln, so daß in den PEs beliebige h-Funktionen implementiert werden können. Dieser Vorteil wird dann deutlich, wenn es um die Implementierung unterschiedlicher h-Funktionen geht. Beim planaren systolischen Feld wäre es zwingend erforderlich, die Multiplikation von Gewichten und Aktivitäten mit anschließender Addition in einem Takt durchzuführen, während bei der neuen Architektur diese Einschränkung entfällt. Noch deutlicher wird der Vorteil, wenn man die Berechnung des euklidischen Abstandes (Subtraktion und Quadrieren und Addition) betrachtet. Insbesondere dann, wenn es nicht möglich ist, auf der Basis eines Full-Custom-Entwurfs die Arithmetikeinheiten zu optimieren, würde beim planaren systolischen Feld die Latenzzeit einer h-Funktion (gewichtete Summe bzw. euklidischer Abstand) die Gesamtleistung erheblich verringern, während bei der neuen Architektur die Latenzzeit in Bezug auf den Datendurchsatz überhaupt keine Rolle spielt.

4.3.5 Optimierung der Anzahl paralleler Prozeßelemente

In den wenigsten Fällen wird die Größe der Gewichtsmatrix, d.h. die Anzahl der Zeilen (m) von $\underline{W}$ der Anzahl paralleler Prozeßelemente (k) auf dem Chip entsprechen. Dies bedeutet, daß nicht genug Prozeßelemente zur Verfügung stehen, um z.B. die Matrix-Matrix-Multiplikation vollständig zu parallelisieren, oder es sind mehr Prozeßelemente vorhanden als erforderlich.

Die Frage, wie viele Prozeßelemente parallel auf einem Chip integriert werden können, wird in erster Linie durch die Zieltechnologie und die maximal zur Verfügung stehende Chipfläche begrenzt. Gerade beim Gate-Array-Design kann die Erhöhung der Anzahl paralleler Prozeßelemente den Wechsel auf einen Wafer mit mehr Zellen je Chip und damit weniger Chips je Wafer (bei gleicher Technologie) bedeuten, was mit einem erheblichen Kostenzuwachs verbunden ist. Daher sollte sehr sorgfältig geprüft werden, wie viele parallele Prozeßelemente auf einem Chip integriert werden sollen. Für eine grobe Abschätzung dieser Frage kann die Effizienz der Parallelisierung herangezogen werden, mit deren Hilfe beurteilt werden kann, in welchem Bereich eine wirtschaftlich sinnvolle Anzahl paralleler Prozeßelemente liegt. Hierfür sind noch einige Randbedingungen bezüglich der Netzgrößen festzulegen. Für die folgenden Betrachtungen soll die maximale Anzahl von Neuronen in einer Schicht auf 512 begrenzt sein, was im Hinblick auf die Praxis für die meisten Anwendungen mehr als ausreichend sein sollte.

Unmittelbaren Einfluß auf die Effizienz hat der Effekt der Partitionierung, d.h. die Abbildung einer Schicht eines neuronalen Netzes auf eine Parallelprozessoranordnung, die anhand der Multiplikation im folgenden untersucht werden soll.

Die Abbildung der Multiplikation zweier Matrizen $\underline{W}$ (m x n) und $\underline{O}$ (n x k) mit $m > k$ auf die neue systolische Kette erfolgt durch Partitionierung der Gewichtsmatrix in Submatrizen der Größe (k x n), so daß die gesamte Matrix-Multiplikation in

$$S = \left\lceil \frac{m}{k} \right\rceil \qquad m, k \in \mathbf{N} \tag{4.20}$$

Schritte zerlegt wird.

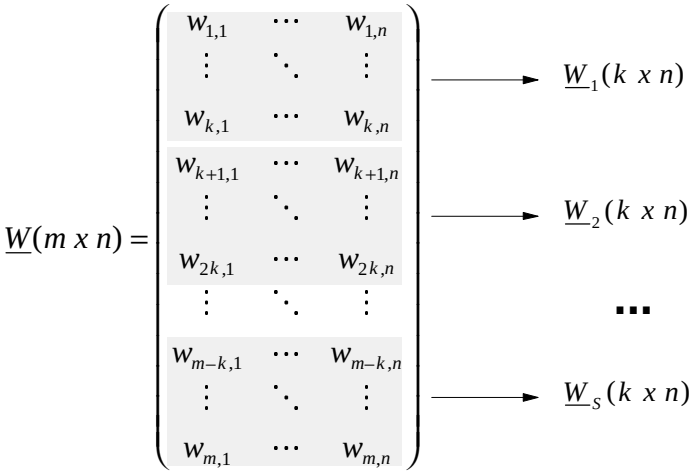


Abbildung 4.10: Partitionierung der Gewichtsmatrix W(m x n) in S Submatrizen der Größe (k x n)

Dabei sind während (S - *1*) Takten alle Prozeßelemente voll ausgelastet. Falls $m \neq S \cdot k$ ist, werden im letzten Schritt nur noch m mod(k) Prozeßelemente benötigt, im schlechtesten Fall also noch eines. Aus der Effizienz, unter Berücksichtigung der Partitionierung,

$$E = \frac{m}{k \cdot \left\lceil \frac{m}{k} \right\rceil} \tag{4.21}$$

läßt sich unter der Voraussetzung

$$m \; mod(k) = 1 \tag{4.22}$$

eine Abschätzung nach unten (E^{MIN}) vornehmen. Unter der Voraussetzung (4.22) gilt:

$$m = (S-1) \cdot k + 1 \tag{4.23}$$

so daß sich für die Effizienz als untere Grenze

$$E^{\min}_{|m \, mod(k)=1} = \frac{m}{m+k-1} \tag{4.24}$$

ergibt, welche bei konstantem m um so kleiner ist, je größer k wird. Das bedeutet, daß infolge der Partitionierung die minimale Effizienz um so geringer ausfällt, je mehr Prozeßelemente vorhanden sind. Der Verlauf der Effizienz, aufgetragen über der Anzahl zu berechnender Neuronen (m), ist für 4, 16 und 64 parallele Prozeßelemente in der Abbildung 4.11 dargestellt.

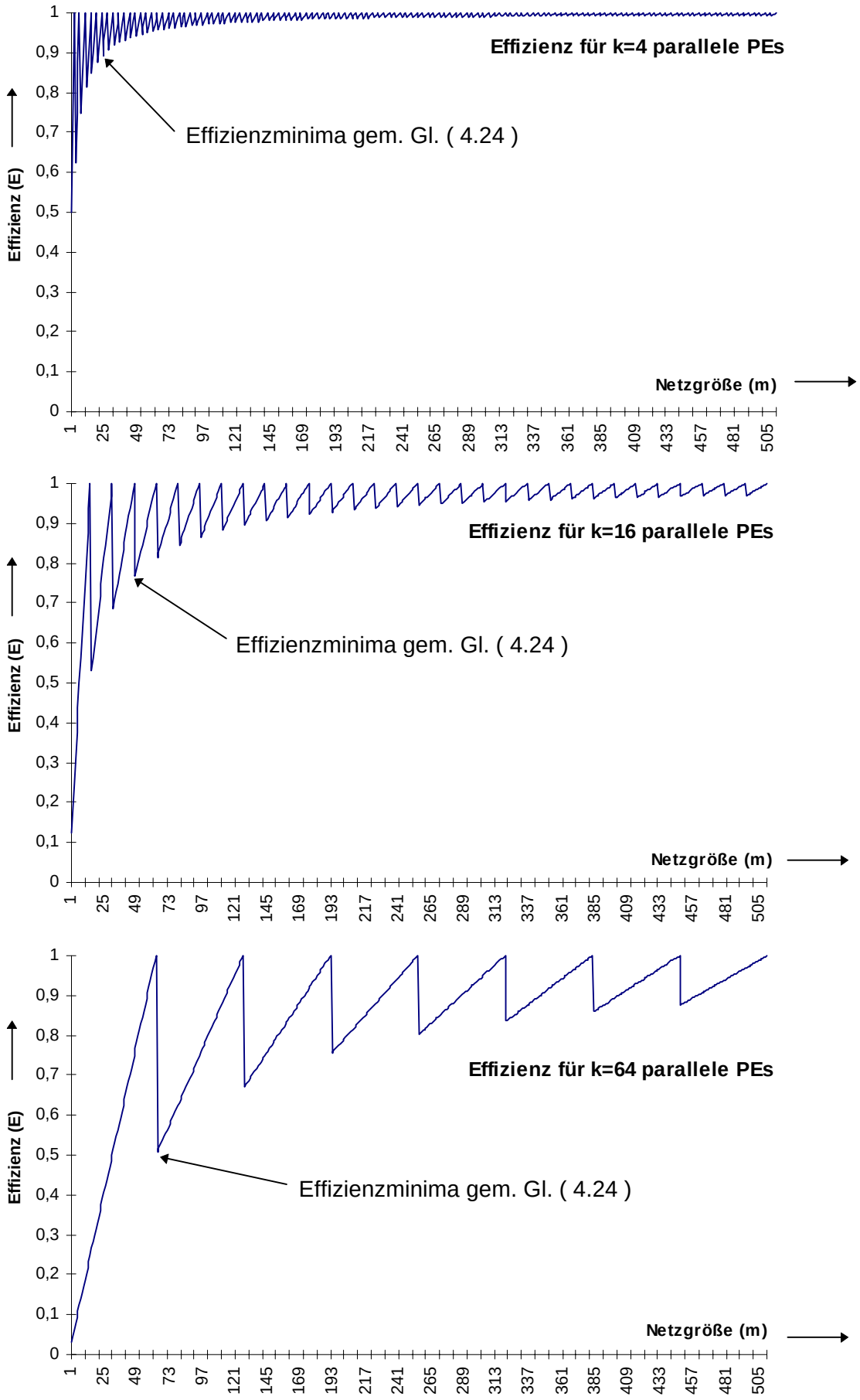


Abbildung 4.11: Verlauf der Effizienz für k=4, 16 und 64 PEs

Diese Tendenz läßt sich auch aus der mittleren Effizienz erkennen. Für Schichten bis 512 Neuronen ($m \leq 512$) gilt:

$$\tilde{E}(k) = \frac{1}{k} \cdot \frac{1}{512} \cdot \sum_{m=1}^{512} \left\lceil \frac{m}{\left\lceil \frac{m}{k} \right\rceil} \right\rceil \qquad (4.25)$$

Fordert man im Mittel z.B. mindestens 90% Effizienz, so läßt sich dies nur erfüllen, wenn nicht mehr als 29 parallele Prozeßelemente verwendet werden.

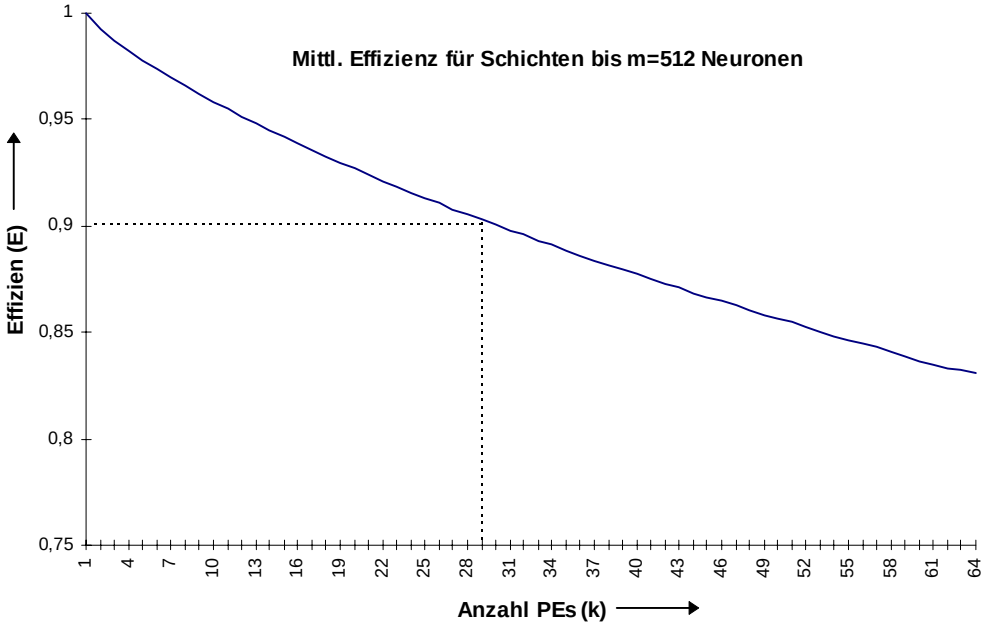


Abbildung 4.12: Mittlere Effizienz für Schichten bis m=512 Neuronen

Bei den bisherigen Überlegungen blieben bei der Beschleunigung und der Effizienz technologische Gesichtspunkte unberücksichtigt. Bedingt durch den Broadcast-Bus (Gewichte), der alle Prozeßelemente verbindet, verursacht jedes weitere Prozeßelement eine zusätzliche kapazitive Last, die sich sicherlich negativ auf die maximale Taktfrequenz auswirken kann. Dieser Einfluß kann allgemein mit einem Faktor $0 \leq \alpha(k) \leq 1$ beschrieben werden, so daß sich für die effektive Maximalbeschleunigung

$$b_{MAX,eff} = \alpha(k) \cdot k \qquad (4.26)$$

ergibt. Dieser Effekt hat zur Folge, daß eine Schicht mit m Neuronen auf einem „kleinen“ Chip mit k_1 parallelen Prozeßelementen unter Umständen schneller berechnet werden kann als auf einem „großen“ Chip mit $k_2 > k_1$ Prozeßelementen, da der kleinere Chip höher getaktet werden kann.

Sowohl die Betrachtungen zur Effizienz als auch zur effektiven Maximalbeschleunigung legen den Schluß nahe, daß es keinesfalls sinnvoll ist, bei der hier vorgestellten neuen sy-

stolischen Kette möglichst viele parallele Prozeßelemente auf einem Chip zu integrieren. Während sich die Frage der Effizienz unabhängig von der Technologie klären läßt, sind für die maximale Beschleunigung (4.26) genauere Betrachtungen bezüglich der Zieltechnologie notwendig.

4.3.6 Vergleich und Bewertung

In den vorausgehenden Abschnitten wurde erläutert, wie sich aus der ursprünglichen systolischen Kette mit lokalen Speichern eine optimierte Variante ableiten läßt. Mit Hilfe der in Kapitel 2.4.2 eingeführten Maßzahlen soll diese neue Architektur bewertet und mit ihren konkurrierenden Versionen Broadcast-Bus und planares systolisches Feld verglichen werden. Diese Betrachtungen werden am Beispiel der Multiplikation zweier Matrizen $\underline{W}$ (m x n) und $\underline{O}$ (n x k) durchgeführt. Neben Effizienz und Beschleunigung, die bereits im Abschnitt 4.3.5 ermittelt wurden, ermöglichen der Datendurchsatz d und die Latenzzeit t_l die objektive Bewertung einer Architektur.

Die Latenzzeit eines Musters der neuen systolischen Kette ergibt sich aus der Anzahl der parallelen Prozeßelemente (k) (welche der Gesamtzahl der zu verarbeitenden Muster entspricht) und der Anzahl der Aktivitäten (n) pro einzulesendem Muster zu

$$t_l^{neu} = k \cdot n + t_{pipeline} . \qquad (4.27)$$

In der additiven Konstanten $t_{pipeline}$ spiegelt sich die Länge der Pipeline eines Prozeßelementes wider. Da bedingt durch die Architektur die Anzahl der simultan zu verarbeitenden Muster mit der Anzahl paralleler Prozeßelemente identisch ist, nimmt mit dem Grad der Parallelisierung die Latenzzeit eines einzelnen Musters zu.

Unabhängig von der Anzahl paralleler Prozeßelemente liest die neue systolische Kette eine Aktivität pro Takt, so daß sich ein Datendurchsatz von einem Datum pro Takt ergibt. Im Vergleich hierzu ist die Latenzzeit eines Musters der Busarchitektur mit One-To-All-Broadcasting t_l^{BC} unabhängig von der Anzahl der zu verarbeitenden Muster, d.h. es gilt:

$$t_l^{BC} = n + t_{pipeline} \qquad (4.28)$$

(n: Anzahl zu verarbeitender Aktivitäten, $t_{pipeline}$: Länge der Pipeline eines Prozeßelements).
Das planare systolische Feld (Variante (a) - Abbildung 2.10) der Größe m x k zur Multiplikation zweier Matrizen $\underline{W}$ (m x n) mit $\underline{O}$ (n x k) besitzt eine Latenzzeit von

$$t_l^{Feld} = n + m - 1 , \qquad (4.29)$$

welche von der Größe des Feldes abhängt. Beim Datendurchsatz ergeben sich folgende Verhältnisse:

Die Architektur mit One-To-All-Broadcasting-Bus hat wie die neue systolische Kette einen Durchsatz von einem Datum pro Takt. Das planare systolische Feld liest innerhalb von n+k-1 Takten $n \cdot k$ Aktivitäten und erreicht dadurch einen Durchsatz von

$$d_{Feld} = \frac{n \cdot k}{n + k - 1}. \qquad (4.30)$$

Wie bereits im vorherigen Abschnitt deutlich wurde, werden sowohl die Effizienz als auch die Beschleunigung erheblich von der Partitionierung beeinflußt. Für die Architektur mit One-To-All-Broadcasting ergeben sich für beide Größen dieselben Verhältnisse wie bei der neuen systolischen Kette, da ebenfalls eine eindimensionale Prozessoranordnung vorliegt. Beim planaren systolischen Feld sind mehrere Varianten denkbar. Zur Vereinfachung wird folgende Annahme getroffen:
Die Anzahl der Prozeßelemente in Ost-West-Richtung (k_0) entspricht der Anzahl der zu verarbeitenden Muster (k), so daß sich das Partitionierungsproblem auf die Netztopologie, d.h. die Abbildung der Gewichtsmatrix (Anzahl der Zeilen m) auf die Anzahl der Prozeßelemente in Nord-Süd-Richtung (m_0) beschränkt. Das Lesen der Ergebnisse nach jedem Schritt wird durch die Zeit t_{read} berücksichtigt. Somit ergeben sich für die Beschleunigung

$$b_{Feld} = \frac{n \cdot m \cdot k}{(n + \max(k, m_0) - 1 + t_{read}) \cdot \left\lceil \frac{m}{m_0} \right\rceil} \qquad (4.31)$$

und für die Effizienz

$$E_{Feld} = \frac{b_{Feld}}{m_0 \cdot k}. \qquad (4.32)$$

Bei der Effizienz ist zu beachten, daß im Gegensatz zu den beiden eindimensionalen Prozessoranordnungen der Ressourceneinsatz nun mit $m_o \cdot k$ zunimmt, was im Falle eines großen Feldes und schlechter Ausnutzung (m mod (m_o)=1) einen erheblichen Effizienzverlust zur Folge hat.

In der folgenden Tabelle sind alle ermittelten Größen nochmals dargestellt.

	Neue systolische Kette	Broadcast-Bus	Planares systolisches Feld
Beschleunigung	$b = \frac{m}{\left\lceil \frac{m}{k} \right\rceil}$		$b = \frac{n \cdot m \cdot k}{(n + \max(k, m_0) - 1 + t_{read}) \cdot \left\lceil \frac{m}{m_0} \right\rceil}$
Effizienz	$E = \frac{b}{k}$		$E = \frac{b}{m_0 \cdot k}$
Datendurchsatz [Worte / Takt]	d =1		$d = \frac{n \cdot k}{n + k - 1}$
Latenzzeit [Takte]	$t_l = k \cdot n + t_{pipeline}$	$t_l = n + t_{pipeline}$	$t_l = n + m - 1$
Anzahl der Busse	3	m+2	m+ k

Tabelle 4.2: Bewertung der neuen systolischen Kette, der Architektur mit Broadcast-Bus (jeweils m Prozeßelemente) und eines planaren systolischen Feldes ($m_0 \cdot k$ Prozeßelemente) für eine m x n Gewichtsmatrix und n Aktivitäten.

Die neue systolische Kette und die Broadcast-Bus-Architektur unterscheiden sich bei den Leistungsdaten nur in der Latenzzeit, welche bei der neuen Architektur von der Anzahl der parallelen Prozeßelemente und damit von der Anzahl simultan zu verarbeitender Muster abhängt. Dies stellt bei Echtzeitanwendungen eine Einschränkung dar, weil mit zunehmender Parallelisierung die Antwortzeiten größer werden. Auf der anderen Seite können so lokale Speicher auf dem Chip bzw. zahlreiche Busse zu externen Speichern vermieden werden. Das planare systolische Feld hat durch seine parallelen Busse den Vorteil, daß es einen hohen Datendurchsatz erreicht, auf der anderen Seite sind auch hier wie bei der Broadcast-Architektur entweder lokale Speicher auf dem Chip oder zahlreiche Busse zu externen Speichern erforderlich. Die geringe Effizienz bei schlechter Auslastung des Feldes ist hier als gravierender Nachteil zu erwähnen, ebenso wie der wesentlich größere Hardwareaufwand des Feldes ($m_0 \cdot k$ Prozeßelemente) im Vergleich zu den beiden anderen Architekturen (k Prozeßelemente).

4.4 Architektur einer kompakten ALU

Das Resultat der bisherigen Betrachtungen ist eine Kommunikationsstruktur, die sich sowohl systolische als auch konventionelle Merkmale einer Busarchitektur zunutze macht und damit die Implementierung unterschiedlich komplexer h-Funktionen ermöglicht. Einzige Randbedingung beim Entwurf eines Prozeßelementes ist die Forderung, daß in jedem Takt ein Aktivitätswert und bei k parallelen Prozeßelementen alle k Takte ein neues Gewicht pro Prozeßelement verarbeitet werden müssen.

Im folgenden Abschnitt wird die Architektur einer kompakten und schnellen ALU als Kernstück eines Prozeßelements vorgestellt, die den Bedürfnissen einer Gate-Array-Implementierung gerecht wird.

4.4.1 Konzeptionelle Betrachtungen zur Architektur der ALU

Zur Implementierung von Funktionen in der Art von Gleichung (4.17) sind verschiedene Konzepte vorstellbar. Um eine geeignete Implementierung zu finden, ist es ratsam, zunächst einige Anforderungen an die ALU zu formulieren.

Anforderungen bedingt durch das Kommunikationsschema der neuen systolischen Kette

Als Folge des reduzierten Kommunikationsaufwandes hat sich eine überlappende Bearbeitung mehrerer Muster ergeben, wobei die Anzahl der innerhalb eines Zyklus zu verarbeitenden Muster der Anzahl paralleler Prozeßelemente in der systolischen Kette entsprechen muß. Das bedeutet, daß im Falle k paralleler Prozeßelemente innerhalb einer ALU k Akkumulationspfade zu implementieren sind. Aufgrund des Zeitmultiplexverfahrens ist es nicht notwendig, diese Pfade vollständig zu parallelisieren, jedoch muß für jedes Muster ein separater Akkumulationsspeicher vorgesehen werden (4.33).

$$\left.\begin{aligned} \Sigma_1 &\leftarrow h\left(w_{ji}, o_{i1}\right) \\ \Sigma_2 &\leftarrow h\left(w_{ji}, o_{i2}\right) \\ &\vdots \\ \Sigma_k &\leftarrow h\left(w_{ji}, o_{ik}\right) \end{aligned}\right\} \quad \forall i \in [1,n] \qquad (4.33)$$

Als weitere Folge der systolischen Verkettung ergibt sich die Notwendigkeit, pro Takt eine Aktivität in der ALU zu verarbeiten. Da die Latenzzeit einer ALU keinen Einfluß auf die Kommunikationszeit hat, kann daher eine Pipelinestruktur für die ALU vorgesehen werden, um hierdurch unterschiedlich komplexe h-Funktionen implementieren zu können.

Anforderungen an Flexibilität und Handhabung

Die Architektur der ALU sollte so flexibel sein, daß möglichst viele der in Kapitel 4.2 dargestellten Algorithmen implementiert werden können, wobei es nicht notwendig ist, der ALU einen General-Purpose-Charakter zu geben um beliebige Algorithmen zu programmieren. Dies ist allein schon wegen der Einschränkung aufgrund der systolischen Kommunikation kaum möglich.

Die Struktur der ALU sollte, soweit dies möglich ist, eine einfache Handhabung aus Sicht des Software-Entwicklers ermöglichen. Neben der Partitionierung betrifft dies in erster Linie die Programmierung verschiedener h-Funktionen.

Anforderungen aufgrund der Gate-Array-Technologie

Die Implementierung auf Basis eines Gate-Arrays hat gegenüber einem Full-Custom-Entwurf den entscheidenden Nachteil, daß keine (hand-) optimierten Arithmetikzellen wie z.B. Multiplizierer oder Addierer verwendet werden können. Statt dessen ist man beim Entwurf auf die Standardzellbibliothek des Herstellers angewiesen. Erweiterungen sind nur auf der Grundlage einer Basiszelle der jeweiligen Technologie möglich, wobei hierfür vom Hersteller entsprechende Entwicklungswerkzeuge zur Verfügung gestellt werden müssen. Im Falle der Gate-Forest-Bibliothek des IMS stand ein Multiplizier-Generator zur Verfügung,

der die automatische Generierung von Multiplizierern im Festkommaformat erlaubt. Wegen des großen Flächenverbrauchs von Gleitkommaeinheiten scheidet diese Form der Verarbeitung aus, wenngleich sie entscheidende Vorteile bezüglich Dynamik und Genauigkeit gegenüber der Festkommadarstellung bietet.

Die Analyse dieser Anforderungen macht deutlich, daß eine ALU in konventioneller „von Neumann Bauart“ weder geeignet noch notwendig ist. Anstelle von Programmen müssen lediglich unterschiedlich komplexe Operationsfolgen ausgeführt werden. Es existieren weder Verzweigungen noch muß eine ALU auf Interrupts reagieren. Aufgrund der Identität aller Prozeßelemente genügt ein globales Steuerwerk (SIMD), so daß lokale Kontrolleinheiten entfallen können.

Als mögliche Architektur bieten sich zwei Varianten an:

- eine **programmierbare** ALU mit frei adressierbaren Registern, wie sie z.B. beim CNAPS 10xx zum Einsatz kommt, oder
- eine **konfigurierbare** ALU, die für die Ausführung bestimmter Operationen optimiert ist.

Für die erste Variante spricht eindeutig die große Flexibilität, die es dem Anwender erlaubt, auch völlig neue Neuronenmodelle oder verwandte Funktionen zu implementieren. Schwierig wird hierbei sein, sämtliche Anforderungen (eine Aktivität pro Takt, überlappende Akkumulation) mit der möglichen Flexibilität zu vereinen. In jedem Fall müßte sich der Anwender mit hardwarespezifischen Besonderheiten beim Softwareentwurf auseinandersetzen. Eine Alternative bestünde darin, durch intelligente Softwarewerkzeuge den Entwickler zu unterstützen, wie dies bei modernen VLIW-Prozessoren der Fall ist. Der TMS320C62xx [Texas97a], ein Vertreter dieser Prozessorklasse, läßt sich komfortabel in C programmieren, ohne daß der Anwender detaillierte Kenntnisse über die darunterliegende Hardware besitzen muß. Damit erkauft man sich aber einen erheblichen Leistungsverlust, da die automatische Codegenerierung nur einen Bruchteil des Parallelisierungspotentials nutzt[10]. Ein zweiter Nachteil sind der Flächenbedarf und der Energieverbrauch einer solchen flexiblen ALU gegenüber einer optimierten, konfigurierbaren ALU. Diese zweite Variante hat jedoch den Nachteil, daß sie aufgrund ihrer festen Verdrahtung nur für einige wenige Funktionen geeignet ist. Sie hat aber den Vorteil, daß bereits durch die Architektur alle Anforderungen erfüllt werden können, die Handhabung wesentlich einfacher wird und dadurch der Anwender von hardwarerelevanten Einschränkungen entlastet werden kann. Dies scheint auch unter Beachtung der in Kapitel 4.2 gewonnen Erkenntnisse bezüglich der parallelisierbaren Algorithmen sinnvoll zu sein, da mit nur zwei h-Funktionen ein Großteil der gebräuchlichsten Netzwerkmodelle realisiert werden kann.

Die folgenden Betrachtungen werden sich daher mit der Architektur einer schnellen, konfigurierbaren ALU auseinandersetzen. Die folgende Abbildung 4.13 zeigt ein Ablaufdiagramm zur Berechnung von (4.17) unter Berücksichtigung hardwarerelevanter Modifikationen. Dieses Ablaufdiagramm stellt im folgenden die Grundlage für den Entwurf der ALU dar.

[10] Siehe hierzu auch Kapitel 5

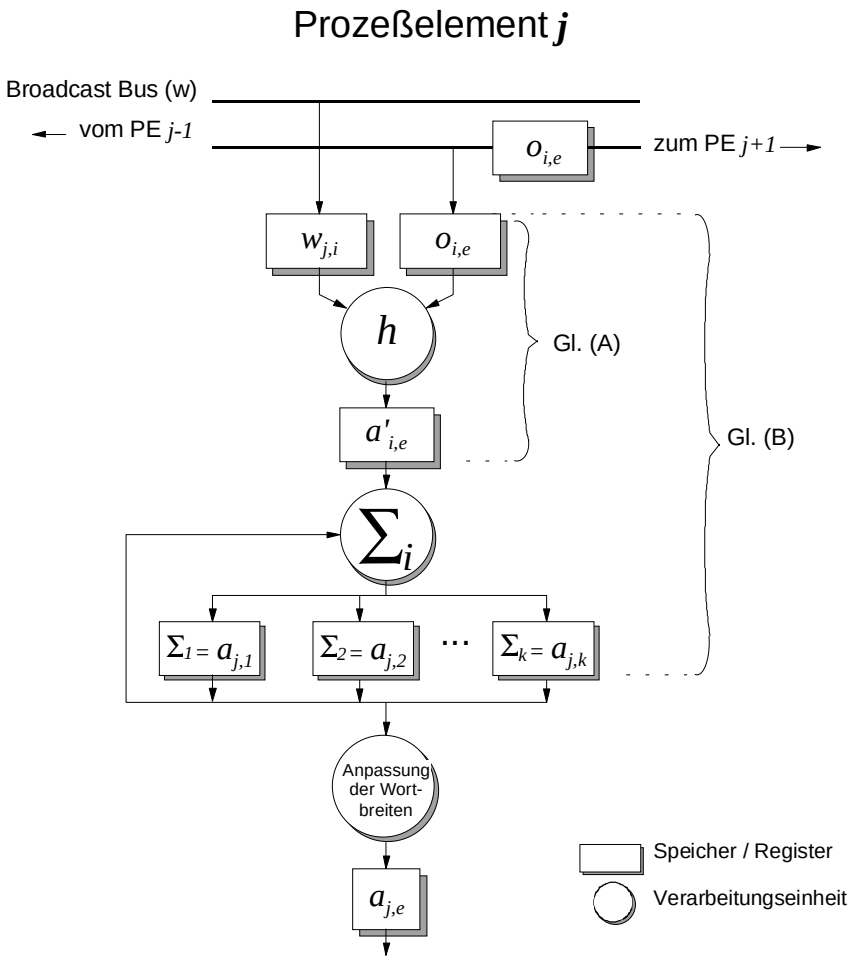


Abbildung 4.13: Ablaufdiagramm zu Gleichung (4.17)

$$a'_{i,e} = h\left(w_{j,i}, o_{i,e}\right) \quad \forall e \in [1,k] \qquad \text{Gl. (A)}$$

$$a_{j,e} = \sum_{i=1}^{n} a'_{i,e} \qquad \forall e \in [1,k] \qquad \text{Gl. (B)}$$

Der strukturelle Aufbau einer ALU besteht aus einem Block (h), der die Verknüpfung der beiden Operanden o und w entsprechend der Konfiguration der ALU durchführt. Diesem Block folgt der Akkumulator mit k Akkumulationsregistern für die überlappende Verarbeitung der Aktivitäten gemäß dem systolischen Prinzip. Die letzte Komponente in der Pipeline führt eine infolge der Festkommaarithmetik notwendige Wortbreitenanpassung der Ausgangsdaten durch, so daß deren Darstellung (Mantisse / Exponent) zu den Eingangsdaten wieder kompatibel ist. Dieser Schritt ist vor allem deshalb notwendig, weil die Ausgangsdaten einer Schicht ν eines Netzes als Eingangsdaten der folgenden Schicht ν+1 dienen.

4.4.2 Arithmetische Funktionen und deren effiziente Implementierung

Der vorangegangene Abschnitt hat die grobe Struktur einer ALU erläutert, ohne dabei im Detail auf die Implementierung der unterschiedlichen h-Funktionen einzugehen. Im Rahmen der Analyse in Kapitel 4.2 wurde deutlich, daß mit nur zwei h-Funktionen, dies sind die Mul-

tiplikation (4.1) und das Quadrieren von Differenzen (4.10), ein wesentlicher Teil der betrachteten Algorithmen abgedeckt werden kann. In Kombination mit einem Akkumulator ergeben sich hierdurch die beiden Funktionen

- Multiply-Accumulate (**MAC**) und
- L2-Norm (**DISTL2**)

Die Abbildung 4.14 zeigt eine Möglichkeit, beide Funktionen bei minimaler Redundanz in Hardware zu implementieren. Während zur Berechnung von (4.1) lediglich ein Multiplizierer benötigt wird, erfordert Gleichung (4.10) einen Subtrahierer und einen Quadrierer. Durch einen umschaltbaren Datenpfad kann der Multiplizierer als Quadrierer für die DISTL2 Funktion genutzt werden, wobei hierfür die Differenz der beiden Operanden w und o auf beide Eingänge des Multiplizierers gelegt wird. Der Subtrahierer kann als Addierer ausgelegt werden, falls die Gewichte mit umgekehrtem Vorzeichen im Speicher stehen. Die Konfiguration der ALU beschränkt sich dann im wesentlichen auf die Steuerung der beiden Multiplexer mux1 und mux2:

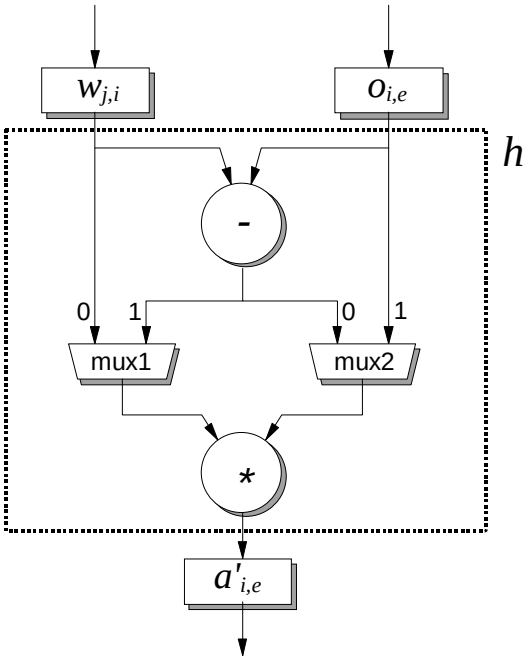


Abbildung 4.14: Hardwarestruktur der h-Funktionen

MUX1	MUX2	Funktion
0	0	undefiniert
0	1	**MAC**
1	0	**DISTL2**
1	1	undefiniert

Wird die DISTL2 Funktion ausgeführt, dann können alle Arithmetikeinheiten der ALU genutzt werden, d.h. die Effizienz beträgt 100%. Bei der MAC-Funktion werden immerhin noch 75% aller Ressourcen einer Alu genutzt. Bei keinem der universell einsetzbaren Neuro-Chips, die im Kapitel 3.2.1 diskutiert wurden ist eine ähnlich hohe Auslastung gewährleistet.

Bei den bisherigen Betrachtungen wurde die Architektur der Arithmetikeinheiten noch nicht berücksichtigt, welche jedoch eine äußerst wichtige Schlüsselrolle im Gesamtentwurf einnimmt. Ein entscheidender Vorteil der im Kapitel 4.4.1 erarbeiteten Kommunikationsstruktur ist die weitgehende Entkopplung der Verarbeitungsdauer innerhalb einer ALU von der Kommunikationszeit in der neuen systolischen Kette. Dies bedeutet, daß es aus Sicht des Kommunikationsnetzwerkes keine Rolle spielt, wie groß die Latenzzeit einer ALU ist. Es muß lediglich gewährleistet sein, daß pro Takt eine Aktivität in jedem Prozeßelement verarbeitet werden kann. Die Dauer einer Taktes bzw. der absolute Datendurchsatz (Anzahl Aktivitäten pro Sekunde) wird lediglich vom kritischen Pfad innerhalb einer ALU begrenzt. Eine effektive Methode, den kritischen Pfad zu verkürzen, ist das Pipelining, d.h. die Unterteilung

einer Operation in zwei oder mehrere Suboperationen [Hennes90]. Am Beispiel einer Multiplizier-Addier-Stufe auf der Basis der IMS-Gate-Forest-Technologie zeigt die Abbildung 4.15 die Verkürzung des kritischen Pfades von $\tau_{kritsich,2} \approx 25ns$ auf $\tau_{kritisch,1} \approx 18ns$ durch Unterteilen in die beiden Basisoperationen „Multiplizieren“ und „Addieren“.

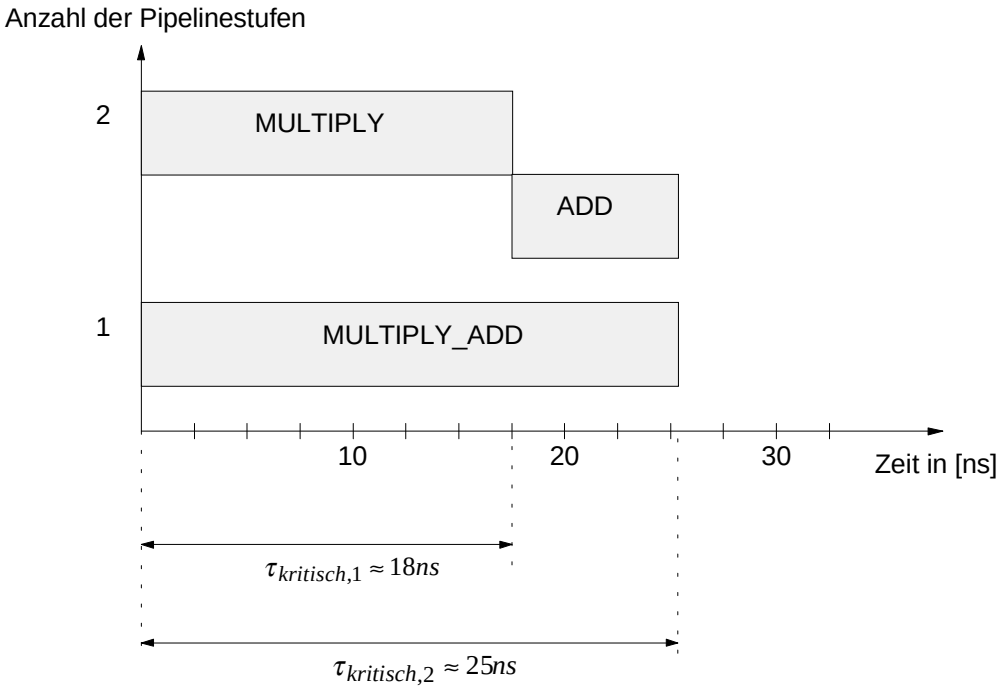


Abbildung 4.15: Verkürzung des kritischen Pfades bei der MAC Operation durch Pipelining am Beispiel der IMS-Gate-Forest-Technologie

Eine weitere Verkürzung des kritischen Pfades wird möglich, wenn eine Basisoperation wie die Multiplikation in weitere Suboperationen unterteilt wird. Man spricht in diesem Fall von Bit-Level-Pipelining [Noll91]. Diese Maßnahmen bewegen sich unterhalb der bisher weitgehend technologieunabhängigen Betrachtungen und erfordern eine genauere Analyse in Frage kommender Multiplizierarchitekturen.

Zum Aufbau von Multiplizier-Arrays mit Pipelinestruktur eignen sich vor allem Carry-Save-Architekturen [Hennes90]. Unabhängig von der gewählten Technologie ist man bemüht, den Flächenbedarf, den Energieverbrauch und die Taktzeit zu reduzieren, so daß der Datendurchsatz maximiert wird. Eine Maßeinheit hierfür ist die Effizienz, welche bereits zur Beurteilung verschiedener Architekturvarianten in Kapitel 2.4.1 eingeführt wurde. Anstelle des Quotienten aus Beschleunigung und Anzahl der eingesetzten Prozessoren wird hier die Effizienz als Quotient aus dem Datendurchsatz und der hierfür notwendigen Chipfläche definiert.

Das Pipelining auf Bitebene hat zwei gegensätzliche Konsequenzen zur Folge: je mehr Pipelinestufen verwendet werden, desto höher wird die erzielbare Maximalfrequenz, da die kritischen Pfade mit zunehmendem Pipelining kürzer werden. Im Grenzfall wird jedes Bit eines Operanden in einer eigenen Pipelinestufe verarbeitet. Durch das massive Pipelining nimmt jedoch die Chipfläche zu, und infolgedessen sinkt die Effizienz.

Wird die Anzahl der Pipelinestufen reduziert, dann sinkt die maximal erzielbare Taktfrequenz durch Verlängerung der kritischen Pfade, andererseits sinkt auch der Flächenverbrauch. In diesem Grenzfall nimmt aber auch die Effizienz durch die sinkende Taktfrequenz wieder ab. Es liegt der Schluß nahe, daß es ein Effizienzoptimum gibt, welches zwischen den beiden Minima (vollständiges Pipelining auf der Ebene von Halbaddierern bzw. kein Pipelining, d.h. vollständig wortparallele Verarbeitung) liegen muß.

Für Multiplizierer mit Carry-Save-Architektur wird die höchste Effizienz erzielt, falls nach zwei Volladdierern jeweils ein Register eingefügt wird [Noll91]. Im folgenden Diagramm sind die Fläche A (normiert auf die Fläche eines Volladdierers), die minimale Taktperiode T_P (normiert auf die Latenzzeit eines Volladdierers) und die Effizienz E über der Anzahl der Volladdierer je Pipelinestufe (v) aufgetragen.

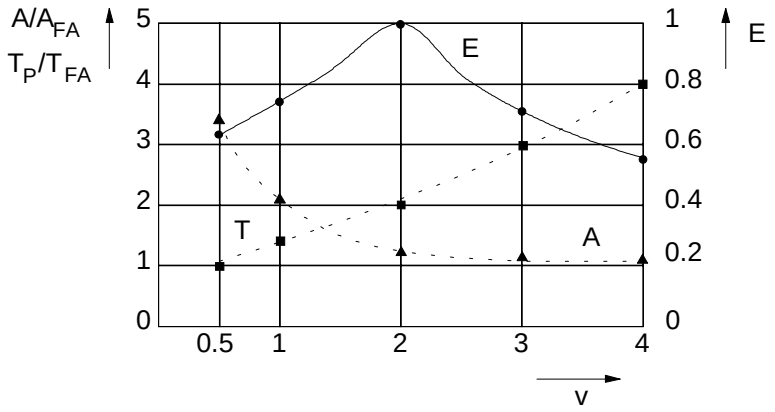


Abbildung 4.16: Verlauf von Fläche (A), Periodendauer (T) und Effizienz (E) als Funktion der Anzahl an Volladdierern je Pipelinestufe (v) [Noll91]

Optimierungen auf dieser Ebene hängen sehr stark von den technologischen Rahmenbedingungen ab und sind häufig nur bei einem Full-Custom-Entwurf möglich.

Neben dieser wortparallelen Verarbeitung gibt es auch bitserielle Multiplizierarchitekturen, die entweder einen Operanden bitseriell und den zweiten wortparallel oder aber beide Operanden bitseriell verarbeiten. Ihr entscheidender Vorteil ist ein deutlich geringerer Flächenverbrauch und eine kürzere Periodendauer gegenüber vollständig wortparallelen Architekturen. Ausschlaggebend für die Leistung einer Architektur ist aber nicht die Taktfrequenz, sondern der maximale Datendurchsatz bzw. die Effizienz.

Läßt die Zellbibliothek nicht die Möglichkeit eines Pipelining auf Bit-Ebene zu, dann stellt sich die Frage, ob der wortparallelen Verarbeitung eine bitserielle vorzuziehen ist. Ungeachtet äußerer Rahmenbedingungen soll die folgende, stark vereinfachte Betrachtung hierüber Aufschluß geben.

Ein wortparalleler Multiplizierer (ohne Bit-Level-Pipelining) liest die beiden Operanden mit einer Frequenz

$$f_{parallel} = f_0 \qquad (4.34)$$

ein. Ein serieller Multiplizierer liest die einzelnen Bits eines Operanden mit der Frequenz

$$f_{seriell,Bit} = r \cdot f_0 \quad r > 1 \qquad (4.35)$$

ein, welche um einen zunächst beliebigen Faktor r größer ist als bei einem Wort-parallelen Multiplizierer. Dies läßt sich damit begründen, daß der serielle Multiplizierer nur 1-Bit-Operationen ausführt und somit schneller getaktet werden kann als der parallele Multiplizierer, der N-Bit Operationen parallel durchführt. Bei einem N-Bit-Wort ergibt sich somit eine Frequenz von

$$f_{seriell,Wort} = \frac{r}{N} \cdot f_0 \quad N \in \mathbf{N} \qquad (4.36)$$

Der Einsatz eines seriellen Multiplizierers lohnt nur dann, wenn für die Flächen der Multiplizierer gilt:

$$A_{parallel} > \frac{N}{r} \cdot A_{seriell} \qquad (4.37)$$

Dann wäre die Effizienz der seriellen Architektur besser als die der parallelen. Im allgemeinen gilt aber eine lineare Skalierung zwischen beiden Architekturen, so daß aus Sicht der Effizienz bzw. des Datendurchsatzes keine eindeutige Entscheidung zugunsten einer der Architekturen getroffen werden kann.

Häufig spielen andere Gründe eine entscheidende Rolle. Sollen möglichst viele Multiplizierer pro Flächeneinheit implementiert werden [Neusser96] oder werden Daten über serielle Verbindungen zwischen verteilten Neuro-Chip-Komponenten übertragen [Ienne94a, Ienne93], dann empfiehlt sich der Einsatz serieller Architekturen. Liegen jedoch beide Operanden in wortparalleler Form vor, so ist eine parallele Architektur vorzuziehen. Der Matrix-Matrix-Multiplizierer des MA16 verwendet beispielsweise Carry-Save-Adder mit Bit-Level-Pipelining [Ramach91].

Ein Pipelining auf Bit-Ebene wird bei der im Rahmen dieser Arbeit erarbeiteten Architektur sicherlich kaum möglich sein, da eine Implementierung auf der Basis einer Gate-Array-Technologie angestrebt wird. Unter Verwendung wortparalleler Rechenwerke und Pipelining auf Wortebene ergibt sich die gesamte Architektur einer ALU gemäß Abbildung 4.17.

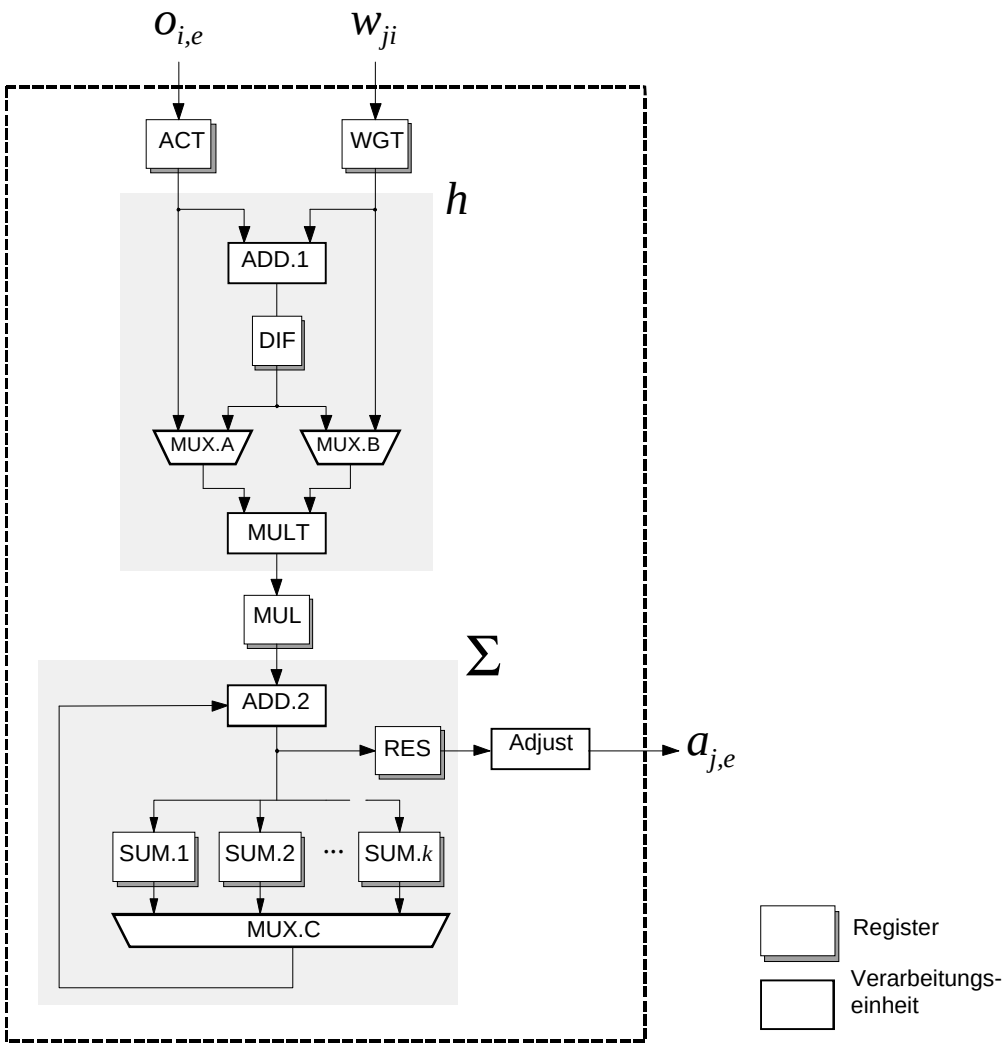


Abbildung 4.17: Architektur einer ALU

4.4.3 Anforderungen an Dynamik und Genauigkeit

Die bisherigen Betrachtungen haben gezeigt, welche Operationen neuronaler Netze und der Bildverarbeitung parallelisierbar sind, bei welchen sich die parallele Implementierung lohnt und wie schließlich die hierfür erforderliche Architektur aussehen kann. Aufgrund der technologischen Randbedingungen wurde die Festkommaarithmetik der Fließkommaarithmetik vorgezogen, welche jedoch in bezug auf Dynamik und Genauigkeit Nachteile besitzt. Die Unterteilung in Mantisse und Exponent bei Fließkommaarithmetik ermöglicht unabhängig von der Operation die Verwendung von Registern gleicher Größe, erfordert aber nach jedem arithmetischen Bearbeitungsschritt eine Normalisierung. Operationen im Festkommaformat erfordern keine Normalisierung, die Operanden besitzen jedoch keine einheitliche Größe, da diese von der durchgeführten Operation abhängt. Eine kritische Betrachtung der erforderlichen Operandengrößen ist unbedingt notwendig, um einerseits die gewünschten arithmetischen Eigenschaften zu gewährleisten, andererseits jedoch die erforderliche Chipfläche so klein wie möglich zu halten. In den folgenden Abschnitten werden daher getrennt für die Recall- und die Lernphase Anforderungen an die erforderliche Dynamik und Auflösung der Gewichte und Aktivitäten neuronaler Netze für die Implementierung in digitaler Hardware untersucht.

Dynamik und Genauigkeit während der Recall-Phase

Approximationsprobleme gelten als die schwierigsten Aufgaben für neuronale Netze, da es nicht, wie bei einfachen Klassifikationsaufgaben, ausreicht, eine Trennlinie zwischen Klassen zu legen. Ziel ist es, einen Kurvenverlauf so exakt wie möglich nachzubilden.

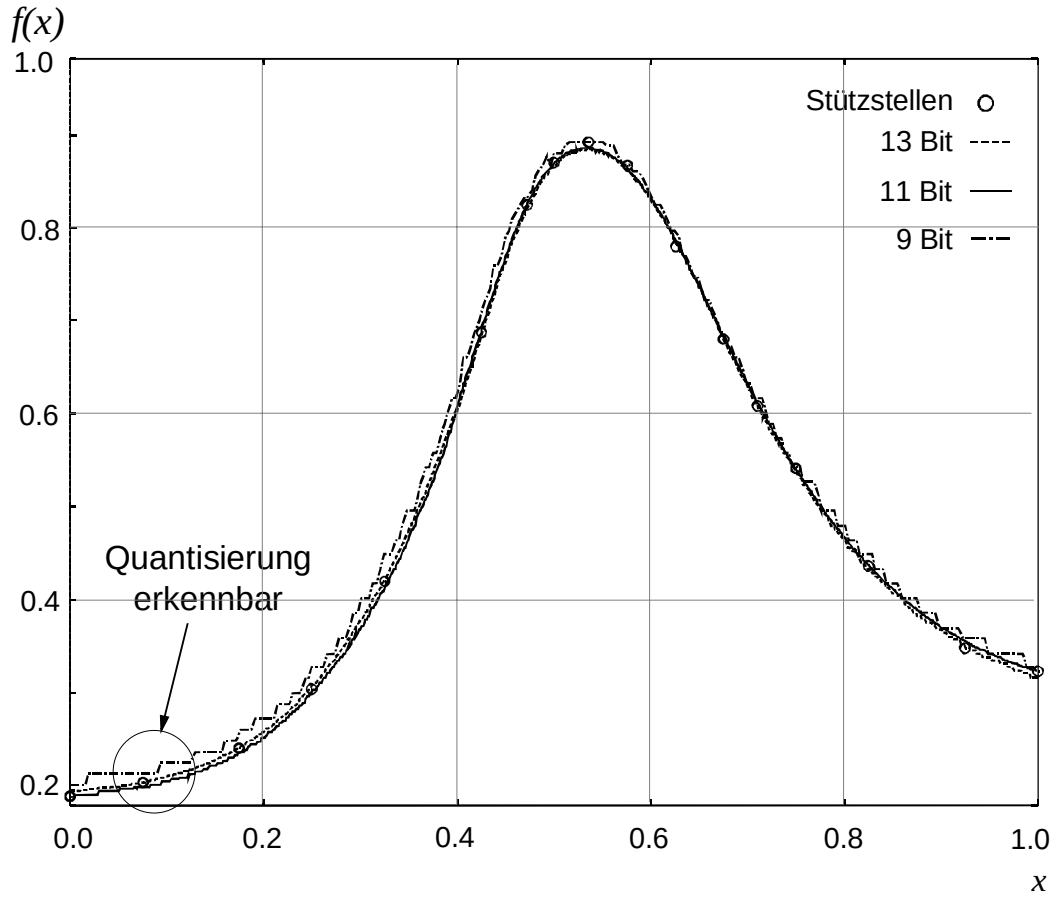


Abbildung 4.18: Recall-Phase für 9, 11 und 13 Bit Wortbreite

Derartige Anforderungen findet man zum Beispiel in der Regelungstechnik, falls ein neuronales Netz die Systemidentifikation durchführen soll [Eppler96]. Zahlreiche Veröffentlichungen [Baker88, Siggelkow91] belegen, daß sowohl für Aktivitäten als auch für Gewichte 8 Bit im Falle einfacher Klassifikationsaufgaben genügen, für anspruchsvollere Aufgaben aber höhere Auflösungen erforderlich sind. Die Abbildung 4.18 zeigt die Approximation einer Funktion mit Hilfe eines MLP-Netzes. Auf der Basis von 17 Stützstellen wurde das Netz (1-12-1) mit Fließkommaarithmetik trainiert und die Recall-Phase mit verschiedenen Wortbreiten für Gewichte und Aktivitäten (9,11 und 13 Bit) simuliert. Lediglich bei 9 Bit kann man Einflüsse der Quantisierung erkennen.

Unter Berücksichtigung üblicher Wortbreiten (8, 16 oder 32 Bit) scheint eine Dimensionierung der Register und Arithmetikeinheiten auf der Basis von 16-Bit-Worten am sinnvollsten zu sein. Dabei ist zu berücksichtigen, daß alle Werte als vorzeichenbehaftete Größen (im Zweier-Komplement) verarbeitet werden. Auf der Grundlage von Abbildung 4.17 ergeben sich für die Register folgende Größen :

$$N_{DIF} = N_{WGT} = N_{ACT} = 16\ Bit \qquad (4.38)$$

$$N_{MUL} = N_{WGT}^{eff} + N_{ACT}^{eff} + 1 = 31\ Bit \qquad (4.39)$$

$$N_{SUM.e} = N_{RES} = \log_2(n) + N_{MUL} \quad \forall e \in [1,k] \qquad (4.40)$$

Die Größe des Multiplikationsregisters MUL ergibt sich aus der Summe der effektiven Wortbreite (ohne Vorzeichen-Bit) von Gewichten und Aktivitäten und einem Vorzeichenbit. Die Breite der Akkumulationsregister SUM.*e*, $e \in \{1,..,k\}$ und des Ergebnisregisters RES hängt unmittelbar von der Anzahl der zu summierenden Elemente (n) ab, deren Maximalwert in Kapitel 4.3.5 mit 512 angenommen wurde.

Somit ergibt sich für die Akkumulations- und das Ergebnisregister

$$N_{SUM.e} = N_{RES} = 40\ Bit \quad \forall e \in [1,k] \qquad (4.41)$$

Dementsprechend sind die Arithmetikeinheiten auszulegen (Tabelle 4.3).

Arithmetikeinheit		Quelle (Operand 1 / Operand 2)		Ziel
Addition	ADD.1	16 Bit	16 Bit	16 Bit
Multiplikation	MULT	16 Bit	16 Bit	16 Bit
Akkumulation	ADD.2	31 Bit	40 Bit	40 Bit

Tabelle 4.3: Dimensionierung der Arithmetikeinheiten

Dynamik und Genauigkeit während des Lernens

Auf der Basis der neuen Architektur soll im folgenden das Training eines Netzes betrachtet werden. Erfolgt der Lernvorgang mit Festkommaarithmetik, so ergeben sich völlig neue Verhältnisse (Abbildung 4.19). Selbst bei 16 Bit Wortbreite wird die Funktion nur sehr unbefriedigend approximiert, bei 13 Bit ist nicht einmal mehr der Verlauf der ursprünglichen Funktion zu erkennen. Erstaunlich ist, daß für alle dargestellten Wortbreiten das summierte Fehlerquadrat (SSE) relativ klein ist (Tabelle 4.4), was zur Folge hat, daß eine Bewertung des Lernergebnisses bei der Betrachtung verschiedener Wortbreiten nicht allein mit Hilfe des SSE, sondern zusätzlich in grafischer Form, wie z.B. in Abbildung 4.19, erfolgen sollte.

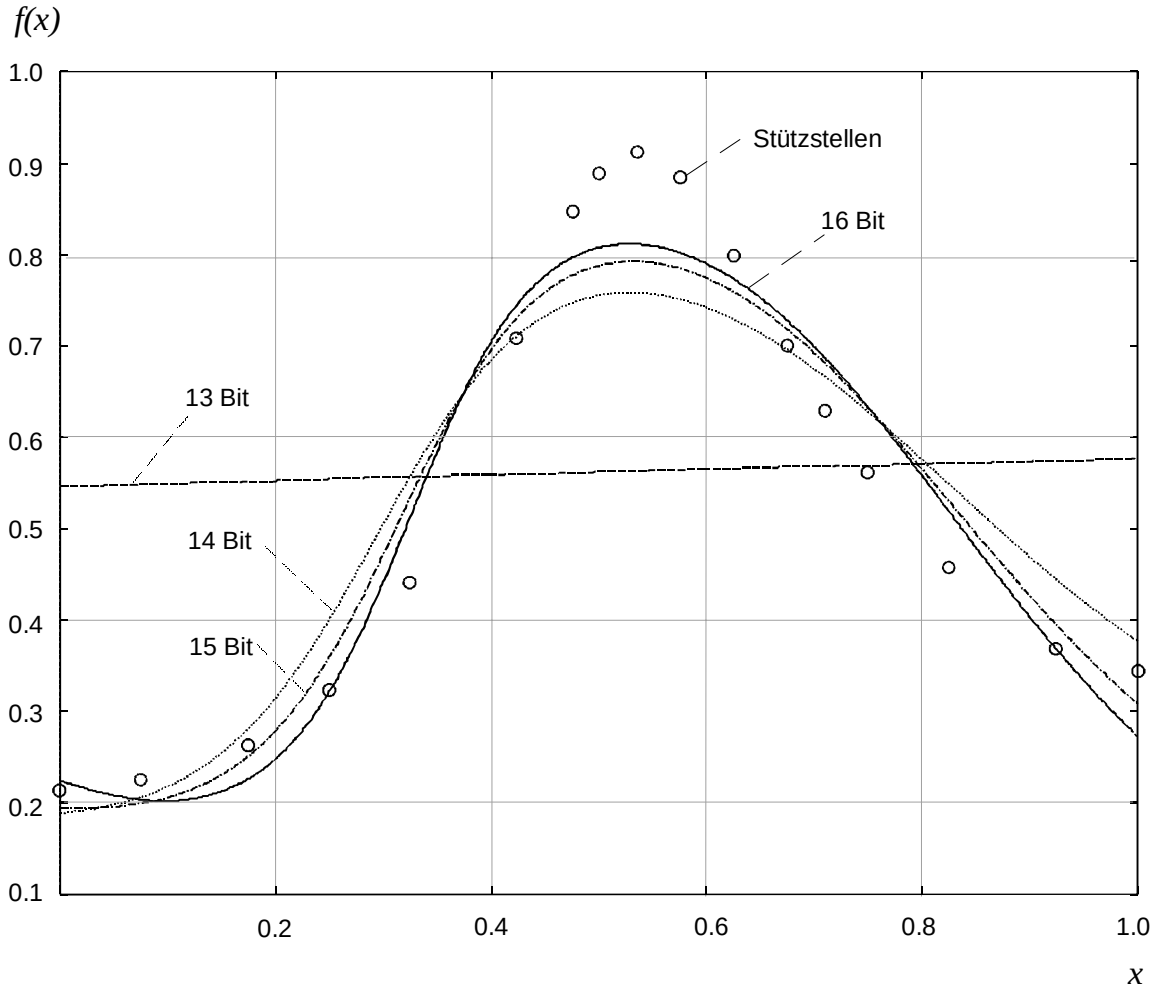


Abbildung 4.19: Trainingsergebnis für unterschiedliche Wortbreiten

Simulierte Wortbreite	Summe der Fehlerquadrate (SSE)	Anzahl der Trainingszyklen
16 Bit	0,053	4164
15 Bit	0,067	3177
14 Bit	0,116	2451
13 Bit	0,98	3

Tabelle 4.4: SSE für unterschiedliche Wortbreiten

Weitere Untersuchungen haben ergeben, daß sich in obigem Beispiel erst ab 24 Bit Wortbreite wesentliche Verbesserungen ergeben. Dies deckt sich auch mit der Untersuchung ähnlicher Probleme von Daniele [Daniele90].

Daraus zu folgern, daß eine Wortbreite von 24 Bit generell notwendig ist, wäre sicherlich falsch, zumal es sich bei der zugrunde liegenden Betrachtung lediglich um ein Beispiel handelt. Es verdeutlicht aber, welche fatalen Folgen eine zu geringe Wortbreite hervorrufen kann. Um dennoch das Training neuronaler Netze mit 16-Bit-Worten realisieren zu können, werden im folgenden Kapitel Architekturerweiterungen zur Beseitigung der oben dargestellten Probleme vorgestellt.

4.5 Erweiterungen der Architektur für das Training neuronaler Netze

Neben dem Vorwärts- und dem Rückwärtsschritt zur Berechnung der Fehler in den Schichten des Netzes, umfaßt das Backpropagation-Lernverfahren einen weiteren Schritt zur Berechnung der neuen Gewichte des Netzes (4.8), (4.9). Dieser Teil des Algorithmus kann nicht durch die systolische Busarchitektur parallelisiert werden und erfordert zusätzliche Hardware. Neben weiteren Registern für die Fehler err_k werden ein Multiplizierer und ein Addierer benötigt. Die eigentliche Problematik des Lernens in Hardware liegt jedoch nicht allein in einer kostengünstigen Implementierung des Algorithmus, sondern vielmehr in den Folgen einer begrenzten Wortbreite (hier 16 Bit). Die Auswirkung der Diskretisierung von Gewichten und Aktivitäten auf das Konvergenzverhalten wurde in der Literatur unter verschiedenen Gesichtspunkten diskutiert [Holt93, Baker88, Roth96]. In den folgenden Abschnitten wird daher der Schwerpunkt auf der Analyse dieser Problematik und der Vorstellung von Lösungsmöglichkeiten liegen.

4.5.1 Verbesserung des Konvergenzverhaltens bei Festkommaarithmetik

Das Beispiel in Kapitel 4.4.3 hat verdeutlicht, daß sich selbst bei einer Wortbreite von 16 Bit für Gewichte und Aktivitäten ein nicht zufriedenstellendes Konvergenzverhalten beim Training eines Netzes ergeben kann. Hierfür können zwei Ursachen identifiziert werden.

Infolge der Diskretisierung der Gewichte kann der n-dimensionale Gewichtsraum als ein n-dimensionales Gitter aufgefaßt werden. Der Abstand der Gitterpunkte entspricht dem kleinsten darstellbaren Zahlenwert auf der Basis der gewählten Diskretisierung. Das schlechte Konvergenzverhalten bei Festkommaarithmetik im Vergleich zur Fließkommaarithmetik ist darauf zurückzuführen, daß die sich theoretisch ergebenden Gewichtsänderungen kleiner als der Gitterabstand sind. Dies hat zur Folge, daß entweder keine Gewichte mehr verändert werden können oder aber infolge von Rundungsfehlern ein Oszillieren entsteht (Abbildung 4.20).

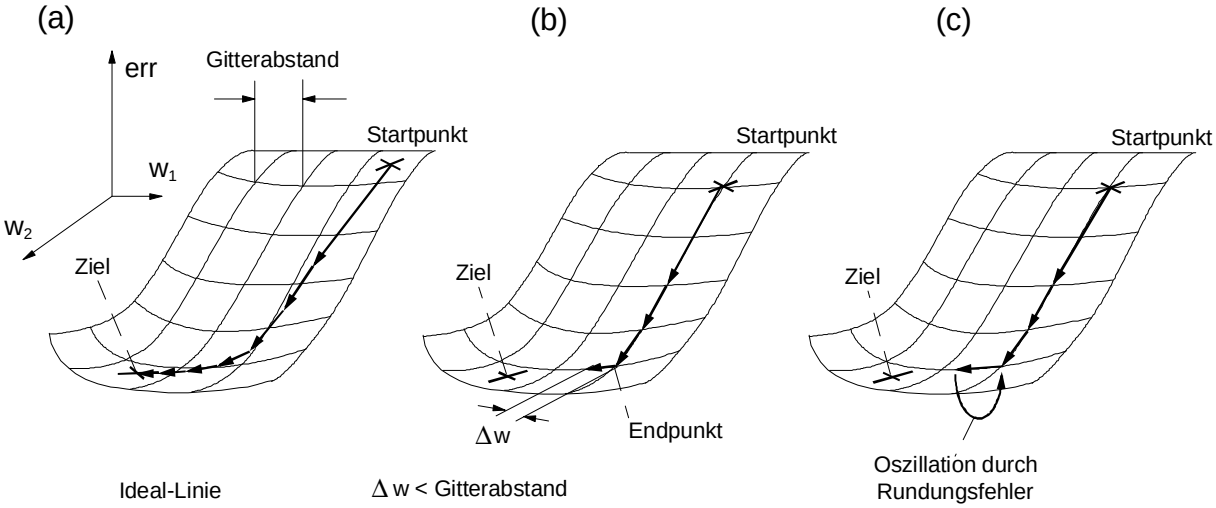


Abbildung 4.20: Idealer Gradientenabstieg (a), Steckenbleiben wegen $\Delta w <$ Gitterabstand (b) und Oszillation wegen Rundungsfehlern (c)

Um zu erreichen, daß das Backpropagation-Lernverfahren infolge der Diskretisierung nicht stecken bleibt, bieten sich verschiedene Lösungsmöglichkeiten an.

Schichtabhängige Lernrate $\eta(\nu)$
In der Regel fallen die Gewichtsänderungen in Schichten, die nahe am Netzausgang liegen, stärker aus als bei Schichten in der Nähe des Netzeingangs. Wird die Lernrate dieser Schichten erhöht, so erreicht man ein homogeneres Training des Netzes, d.h. eine stärkere Einbindung der vorderen Schichten in den Lernprozeß, was ein verbessertes Konvergenzverhalten zur Folge hat.

Verdopplung des Fehlers in der Ausgangsschicht
Vor allem am Ende des Lernvorganges, wenn die Gewichtsänderungen kleiner werden, tritt der oben beschriebene Fall auf, daß ein Gewichtsänderungsvektor zu kurz ist, um den nächsten Gitterpunkt zu erreichen. Eine Möglichkeit, dieses Problem in den Griff zu bekommen, wäre eine Erhöhung sämtlicher Lernraten am Ende des Trainings. Hierbei ist aber zu beachten, daß diese Multiplikation die letzte eines langen Berechnungspfades ist, in den zahlreiche Rundungsfehler einfließen. Diese werden dann durch die größeren Lernraten noch verstärkt. Eine andere Möglichkeit besteht darin, den Fehler am Ausgang des Netzes zu verdoppeln, wodurch letztlich ebenfalls eine Vergrößerung der Gewichtsänderungsvektoren erreicht wird, wobei diese aber wesentlich genauer sind.

Additives Rauschen
Addiert man zu den Gewichtsänderungen mittelwertfreies Rauschen, so kann hierdurch ebenfalls eine Verbesserung des Konvergenzverhaltens erzielt werden [Vincent91]. Durch das Rauschen ist es möglich, einen benachbarten Gitterpunkt auch dann zu erreichen, wenn der Gewichtsänderungsvektor hierfür eigentlich zu kurz wäre. Das mittelwertfreie Rauschen gewährleistet, daß im Mittel derjenige Gitterpunkt erreicht wird, der sich in Richtung des Gewichtsänderungsvektors befindet.

Anhand eines statistischen Modells [Roth96] wurde die Wirkungsweise dieser Erweiterungen untersucht und in Simulationen am Beispiel des Approximationsproblems des Kapitels 4.4.3 überprüft. Vor allem die Kombination aus allen drei Methoden ermöglicht das beste Ergebnis. (Abbildung 4.21).

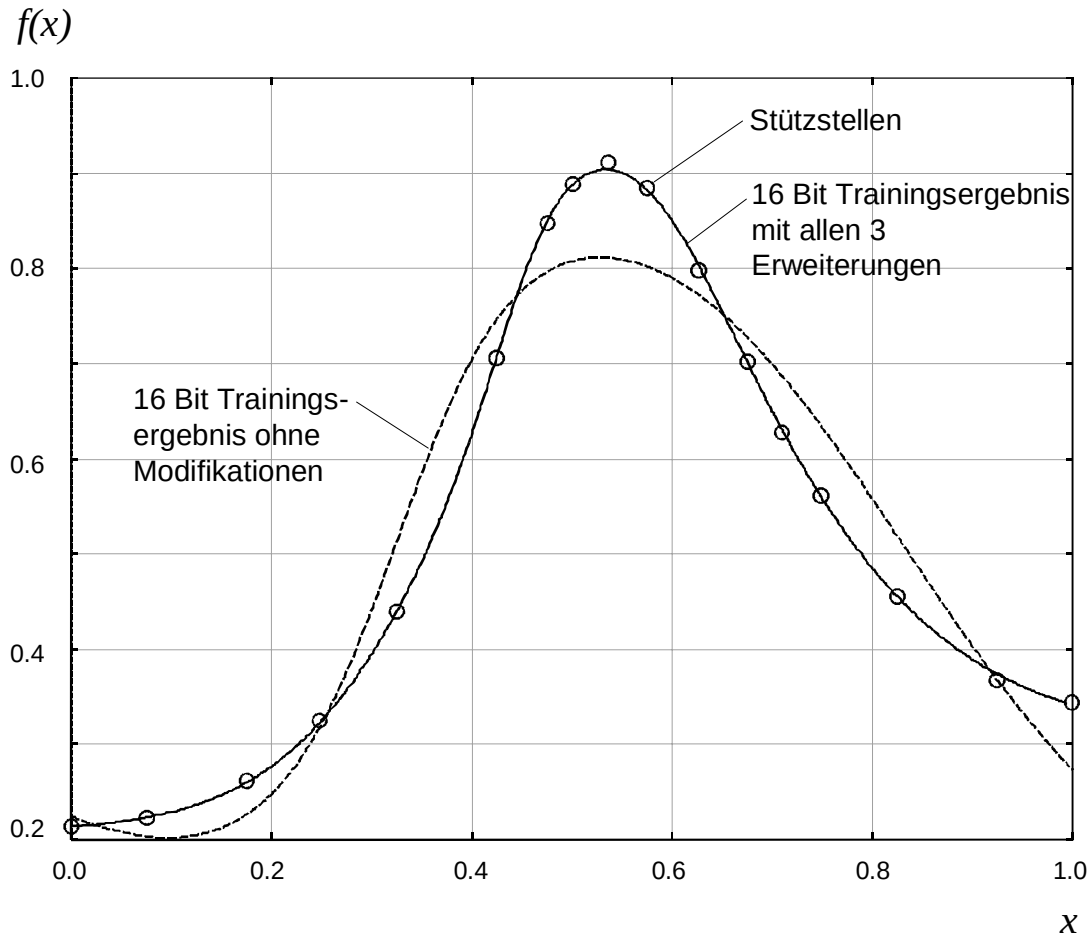


Abbildung 4.21: Ergebnis des Trainings mit allen drei Modifikationen

4.5.2 Optimiertes Rundungsverfahren

Betrachtet man die neue parallele Architektur aus Kapitel 4.3, so sind Rundungsfehler bei der Verringerung der Wortbreite von 40 auf 16 Bit im mit ADJUST (Abbildung 4.17) bezeichneten Block zu erwarten. Im einfachsten Fall würde aus der 40 Bit Summe am Ausgang des Akkumulators ein 16 Bit Fenster herausgeschnitten. Ein Sättigungsmechanismus muß gewährleisten, daß keine Überläufe möglich sind. D.h., ist ein Bit oberhalb des 16 Bit Fensters gesetzt (positive Werte vorausgesetzt), dann muß Sättigung eintreten. Die Behandlung von niederwertigen Nachkommastellen, die im Zuge der Wortbreitenanpassung ebenfalls wegfallen, erfordert ein geeignetes Rundungsverfahren.

Für die folgenden Betrachtungen gelte folgende Notation:
Eine Binärzahl im Festkommaformat mit einem Vorzeichenbit, weiteren N_{VK} Stellen vor dem Komma und N_{NK} Stellen nach dem Komma besitze eine Darstellung gemäß Abbildung 4.22.

Abbildung 4.22: Notation für Binärzahlen im Festkommaformat

Bekannte Rundungsverfahren wie z.B.

- der Cut-Mechanismus (Abschneiden ohne Beachtung der Werte)
- das Jamming (niederwertigstes Bit der neuen Zahl wird dann gesetzt, wenn mindestens eines der abgeschnittenen Bits gesetzt ist)
- das exakte Runden (höchstwertiges Bit der abgeschnittenen Stellen wird zum niederwertigsten Bit der neuen Zahl addiert)

berücksichtigen jedoch nicht die Gegebenheiten beim Runden in einem neuronalen Netz. Betrachtet man das Runden an der oben beschriebenen Stelle in einem größeren Kontext, so wird deutlich, daß Rundungsfehler unmittelbar in die Abbildung der Aktivierungen durch die logistische Funktion einfließen. Untersuchungen belegen [Siggelkow91], daß die Implementierung der logistischen Transferfunktion einen erheblichen Einfluß auf das Konvergenzverhalten eines in Hardware implementierten Netzes hat. Daher liegt es nahe, die Anpassung der Wortbreiten und das Runden in Zusammenhang mit der Implementierung dieser Funktion zu untersuchen. Hierfür sind zunächst einige grundlegende Betrachtungen erforderlich.

Der Definitionsbereich der logistischen Transferfunktion sei gemäß Gleichung (4.42) festgelegt, d.h. Werte darunter oder darüber werden durch Sättigung beschränkt.

$$\mathbf{D}_{Transfer} = \left\{ a_j \mid -16{,}0 \le a_j < 16{,}0 \right\} \qquad (4.42)$$

Dieser Definitionsbereich wird durch die Verringerung der Wortbreite von 40 auf 16 Bit in äquidistante Stücke der Größe

$$\delta_{a_j} = \frac{\left| a_{j,\max} - a_{j,\min} \right|}{2^{16}} = \frac{\left| \frac{2^{15}-1}{2^{11}} - \frac{-2^{15}}{2^{11}} \right|}{2^{16}} \approx 488{,}2738 \cdot 10^{-6} \qquad (4.43)$$

unterteilt. Durch den nichtlinearen Charakter der logistischen Transferfunktion ergibt sich auf der Ordinate eine inhomogene Quantisierung. Die Größe einer Quantisierungseinheit hängt vom Abszissenwert folgendermaßen ab:

$$\delta_{o_j} = \left| f_{sigmoid}^{logistisch}(a_j) - f_{sigmoid}^{logistisch}(a_j + \delta_{a_j}) \right| \quad mit \; f_{sigmoid}^{logistisch} = \frac{1}{1+e^{-b \cdot a_j}} \qquad (4.44)$$

$$\delta_{o_j}(a_j) = \left| \frac{e^{-b \cdot a_j} - e^{-b \cdot \left(a_j + \delta_{a_j} \right)}}{\left(1+e^{-b \cdot a_j} \right) \cdot \left(1+e^{-b \cdot \left(a_j + \delta_{a_j} \right)} \right)} \right| \quad für \; b > 0 \wedge \delta_{a_j} > 0$$

Für verschiedene Parameter b (Formfaktor der logistischen Funktion) ist der Verlauf von $\delta_{o_j}(a_j)$ in Abbildung 4.23 dargestellt.

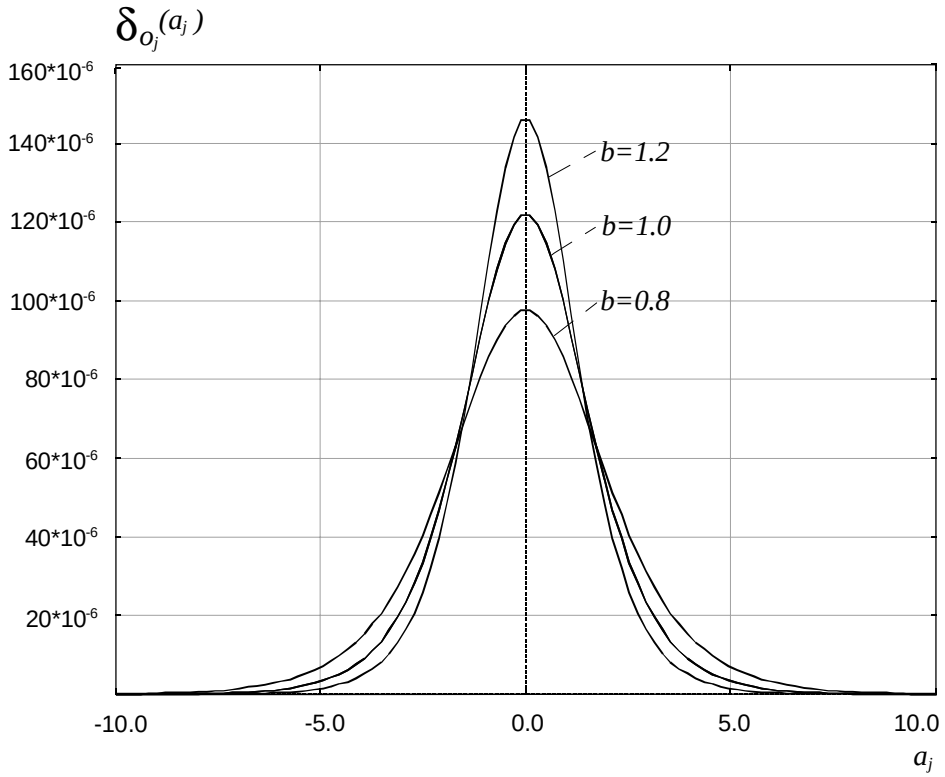


Abbildung 4.23: Quantisierungsverlauf der Ordinate

Der nichtlineare Verlauf der Quantisierung mit Maximum bei a_j=0 läßt sich leicht anschaulich erläutern. Da die erste Ableitung der Sigmoidfunktion keine Konstante ist, werden Intervalle auf der Abszisse auf unterschiedlich große Intervalle auf der Ordinate abgebildet. Sieht man von einer geeigneten Wahl des Parameters b ab, so besteht eine Möglichkeit, den Gesamtfehler zu reduzieren, darin, die Intervalle auf der Abszisse der Steigung der logistischen Funktion anzupassen. Die Frage, wie groß die Intervalle auf der Abszisse zu wählen sind, kann beantwortet werden, wenn das Ziel, ein möglichst geringer Fehler der Ordinatenwerte, betrachtet wird. Hierbei kann man sich an den Eingangsaktivitäten orientieren, deren Definitionsbereich gemäß (4.46) festgelegt ist. Bei 16 Bit Auflösung ergibt sich eine konstante Quantisierung der Größe

$$\delta_{o_j} \approx 30{,}171 \cdot 10^{-6} \qquad (4.45)$$

$$\mathbf{D}_{ActIn} = \{o_i \mid -1{,}0 \le o_i < 1{,}0\} \qquad (4.46)$$

Fordert man auf der Ordinate Intervalle dieser konstanten Größe δ_{o_j}, so kann unter Benutzung der Umkehrfunktion der logistischen Funktion (4.47) eine vom Ordinatenwert o_j abhängige Intervallgröße (b=1) auf der Abszisse angegeben werden (4.48). Um ein Intervall $\delta_{a_j}(o_j)$ vorgegebener Größe darstellen zu können, werden mindestens $N(o_j)$ (4.49) Bit benötigt. Der hieraus resultierende Verlauf von $N(o_j)$ ist in Abbildung 4.24 dargestellt.

$$a_j = \tilde{f}\left(o_j\right) = \frac{1}{b} \cdot \ln\left(\frac{o_j}{1-o_j}\right) \quad \forall\, 0 < o_j < 1 \qquad (4.47)$$

$$\delta_{a_j}\left(o_j\right) = \left| \ln\left(\frac{o_j}{1-o_j}\right) - \ln\left(\frac{o_j + \delta_{o_j}}{1-o_j-\delta_{o_j}}\right) \right| \quad \forall\, 0 < o_j < 1-\delta_{o_j} \;\wedge\; b = 1 \qquad (4.48)$$

$$N\left(o_j\right) \geq -\log_2\left(\delta_{a_j}\left(o_j\right)\right) \qquad (4.49)$$

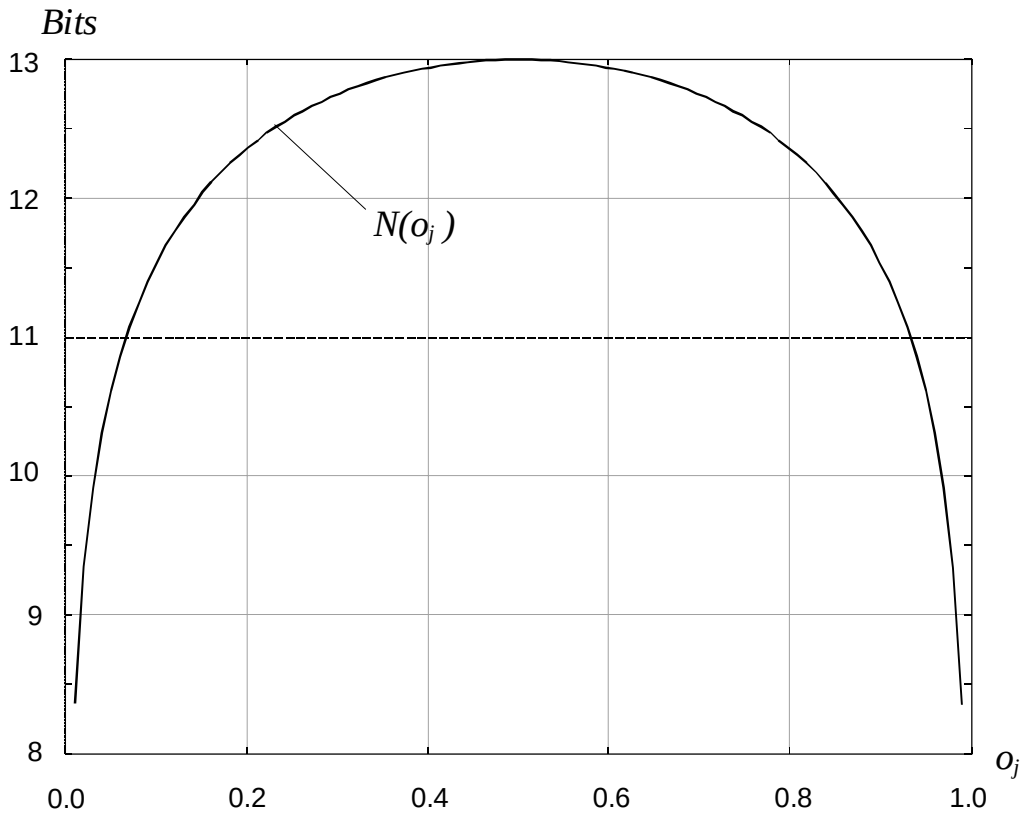


Abbildung 4.24: Anzahl erforderlicher Bit für die Darstellung der Nachkommastellen bei konstanter Auflösung der Ausgangsaktivitäten o_j

Aus dieser Betrachtung (Abbildung 4.24) ist zu erkennen, daß im Punkt der größten Steigung der logistischen Funktion (Gleichung (2.3) bzw. Abbildung 2.3) bei o_j=0,5 mindestens 13 Bit benötigt werden. Bei 16 Bit Auflösung und der ursprünglichen festen Rundung auf 11 Bit (für die Nachkommastellen) wird deutlich, daß ein konventionelles Rundungsverfahren nicht in der Lage wäre, einen Gewinn von zwei Bit zu ermöglichen.

Als Konsequenz dieser Untersuchungen ergibt sich die Notwendigkeit, Aktivierungen am Ausgang des Akkumulators mit unterschiedlicher Präzision zu verwenden. Hält man an einer Gesamtwortbreite von 16 Bit fest, so entspricht die Verwendung unterschiedlicher Präzisionen einer Verschiebung des Dezimalpunktes in der Festkommadarstellung. Die Position des Dezimalpunktes muß dann mit einem geeigneten Verfahren im Wort selbst kodiert werden. Hierfür können maximal 2 Bit benutzt werden, um gerade noch 13 Bit für die höchste Auflösung verwenden zu können. Von den zahlreichen Methoden eine (beliebige) nichtlineare Transferfunktion in Hardware zu implementieren [Schrau98, Siggelkow91], scheint die Verwendung einer Look-Up-Tabelle am universellsten zu sein. Damit können sowohl codierte als auch uncodierte Funktionen implementiert werden.

Im folgenden wird eine mögliche Codierung unterschiedlicher Positionen des Kommas für die logistische Funktion vorgestellt. Das codierte Wort setzt sich aus mehreren Blöcken, in der Tabelle 4.5 durch rechteckige Klammern symbolisiert, zusammen. Der erste Block, z.B. [00] beschreibt die Kennzeichnung, der zweite repräsentiert das Vorzeichen, der dritte die Anzahl der Vorkomma-Bit, z.B. [1X] für ein Vorkomma-Bit. Der letzte Block repräsentiert schließlich die Anzahl der Nachkomma-Bit. Aus Symmetriegründen genügt es, lediglich den positiven Bereich [0,16) zu betrachten.

Bereich	Intervall	Intervallgröße	N_{VK}	N_{NK}	$\delta_{a,xx}$	Codierung
00	[0,1)	$\Delta=1-\delta_{a,00}$	0	13	$122\cdot10^{-6}$	[00] [S] [0X].[13Y]
01	[1,2)	$\Delta=1-\delta_{a,01}$	0	13	$122\cdot10^{-6}$	[01] [S] [0X].[13Y]
10	[2,4)	$\Delta=2-\delta_{a,10}$	1	12	$244\cdot10^{-6}$	[10] [S] [1X].[12Y]
11	[4,16)	$\Delta=12-\delta_{a,11}$	4	9	$1{,}953\cdot10^{-3}$	[11] [S] [4X].[9Y]

Tabelle 4.5: Codierung der Kommaposition

Durch die Separierung des Intervalls auf der Basis von 2er-Potenzen (Tabelle 4.5) ist es nicht möglich, den idealen Verlauf der Auflösung (Abbildung 4.24) nachzubilden, so daß sich eine treppenförmige Annäherung ergibt (Abbildung 4.25). Die resultierende Auflösung liegt aber, bis auf den Bereich 4, oberhalb der geforderten Genauigkeit (schraffierter Bereich).

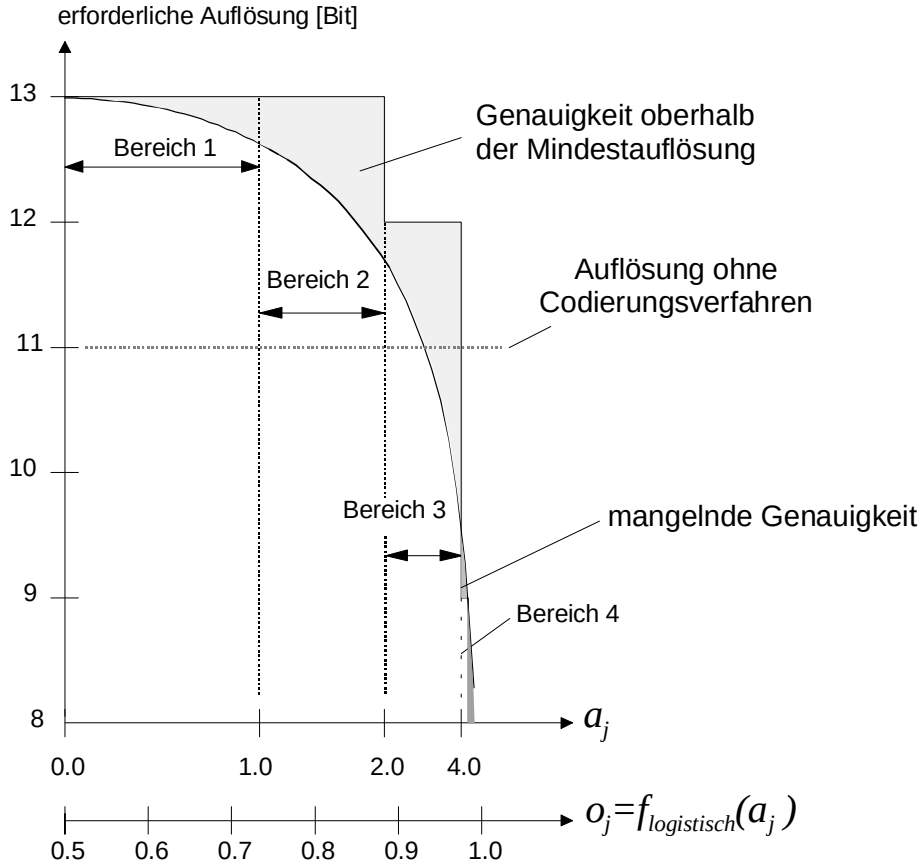


Abbildung 4.25: Anpassung der Quantisierung an die logistische Funktion

Dieses Codierungsschema läßt sich auch auf andere Funktionen anwenden, die bei a_j=0 die größte Steigung (im betrachteten Intervall) besitzen. Hierzu zählt z.B. die Funktion (4.11), die bei der Implementierung von RBF-Netzen verwendet wird. Für alle anderen Transferfunktionen, die nicht von diesem Schema profitieren, sollte daher eines der Standardrundungsverfahren, im einfachsten Fall ein Cut-Mechanismus, vorgesehen werden.

4.5.3 Implementierung in Hardware

Die Betrachtungen des letzten Kapitels haben gezeigt, daß die Verwendung einer Look-Up-Tabelle in Zusammenhang mit einem neuen Codierungsverfahren die arithmetischen Eigenschaften der Festkommadarstellung für häufig verwendete Aktivierungsfunktionen verbessern kann. Da bei 16 Bit Daten die Tabelle eine Größe von 128 KBytes besitzt, scheint eine Implementierung auf dem Chip auszuscheiden. Dies hat zur Folge, daß auch die Zwischenspeicherung von Aktivitäten verdeckter Schichten nicht auf dem Chip implementiert werden muß und statt dessen externe Speicher verwendet werden. Ein damit verbundener Nachteil ist mit Sicherheit die geringere Integrationsdichte.

Die Implementierung des oben beschriebenen Codierungsverfahrens erfordert nur ein Minimum an zusätzlicher Hardware. Im Rahmen der Sättigung müssen ohnehin die Stellen vor dem Komma analysiert werden, so daß lediglich hier zusätzlich ausgewertet werden muß, in welchem Bereich die jeweilige Aktivierung liegt. Hierzu werden (bei positiven Werten) die Stellen des höchstwertigen Bit, welches gesetzt ist, ermittelt. Hierfür genügen Standardkomponenten einer Gate-Array-Bibliothek.

4.6 Skalierung der Architektur

4.6.1 Zielsetzung

In Kapitel 4.3.5 wurde anhand der Effizienz erläutert, daß es keinesfalls sinnvoll ist, generell eine möglichst große Anzahl paralleler Prozeßelemente auf einem Chip zu implementieren. Um dennoch die Anzahl paralleler Prozeßelemente der jeweiligen Aufgabe anzupassen, muß die Architektur skalierbar sein. Um zu erreichen, daß die Latenzzeit dabei konstant bleibt, werden hierfür mehrere, in sich abgeschlossene systolische Ketten parallel geschaltet, wodurch ein SIMD-Array entsteht. Dabei ist zu beachten, daß die parallelen systolischen Ketten auf den gleichen Eingangsdaten operieren. Das folgende Kapitel beschreibt, wie die Gewichtsmatrix im Falle des Parallelbetriebs mehrerer systolischer Ketten partitioniert werden muß.

4.6.2 Partitionierung bei Parallelbetrieb mehrerer Neuro-Chips

Eine Gewichtsmatrix $\underline{W}$ wird auch hier, wie bei der Betrachtung mit nur einem Chip, in S Submatrizen der Größe k=k_{PE}[11] gemäß Gleichung (4.20) zerteilt. Die Submatrizen werden jedoch nicht vollständig sequentiell, sondern, abhängig von der Größe k_{PE} und der Anzahl paralleler Neuro-Chips k_C, parallel in

$$S' = \left\lceil \frac{m}{k_{PE} \cdot k_C} \right\rceil \qquad (4.50)$$

Schritten verarbeitet. Wie bei den Betrachtungen zur Effizienz gilt auch hier, daß eventuell im letzten Schritt S' nur noch ein Teil der parallelen Ressourcen genutzt wird. Im ungünstigsten Fall wird von den k_C parallelen Chips nur noch ein Prozeßelement eines Chips benötigt. Die entsprechenden Zeilen der Gewichtsmatrix sind mit Nullen aufzufüllen.

Für den angepaßten Fall

$$S' = \frac{m}{k_{PE} \cdot k_C} \qquad (4.51)$$

ist die gesamte Partitionierung der Gewichtsmatrix in der folgenden Abbildung 4.26 dargestellt:

[11] k_{PE} gibt die Anzahl paralleler Prozeßelemente je Chip an

$$\underline{W} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{pmatrix} = \left\{ \begin{array}{l} \left. \begin{array}{l} \underline{W}_1 = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k_{PE},1} & \cdots & w_{k_{PE},n} \end{pmatrix} \\ \vdots \\ \underline{W}_{k_C} = \begin{pmatrix} w_{k_C-k_{PE},1} & \cdots & w_{k_C-k_{PE},n} \\ \vdots & \ddots & \vdots \\ w_{k_C,1} & \cdots & w_{k_C,n} \end{pmatrix} \end{array} \right\} \Rightarrow Schritt\ 1 \\ \vdots \\ \left. \begin{array}{l} \underline{W}_{(S'-1)\cdot k_C} = \begin{pmatrix} w_{(S'-1)\cdot k_C-k_{PE},1} & \cdots & w_{(S'-1)\cdot k_C-k_{PE},n} \\ \vdots & \ddots & \vdots \\ w_{(S'-1)\cdot k_C,1} & \cdots & w_{(S'-1)\cdot k_C,n} \end{pmatrix} \\ \vdots \\ \underline{W}_{S'\cdot k_C} = \begin{pmatrix} w_{S'\cdot k_C-k_{PE},1} & \cdots & w_{S'\cdot k_C-k_{PE},n} \\ \vdots & \ddots & \vdots \\ w_{S'\cdot k_C,1} & \cdots & w_{S'\cdot k_C,n} \end{pmatrix} \end{array} \right\} \Rightarrow Schritt\ S' \end{array} \right.$$

Abbildung 4.26: Partitionierung der Gewichtsmatrix bei Parallelbetrieb mehrerer Neuro-Chips

4.6.3 Automatische Partitionierung und Distribution der Gewichtsmatrix

Da die $S' \cdot k_C$ Submatrizen auf insgesamt k_C verteilte Speicher aufgesplittet werden müssen, könnte man vermuten, daß trotz der programmierfreundlichen Architektur der Anwender detaillierte Hardwarekenntnisse besitzen muß, um eine Gewichtsmatrix richtig zu unterteilen und zu speichern. Es läßt sich zeigen, daß mit Hilfe eines einfachen Verfahrens, welches sich leicht in Hardware implementieren läßt, die Partitionierung weitgehend automatisiert werden kann.

Aus der Sicht des Benutzers ist es sicherlich wünschenswert, daß die Anzahl paralleler Chips beim Speichern der Gewichte keine Rolle spielt. Dies ist möglich, wenn die Submatrizen spaltenweise an einen Controller übergeben werden, der die Distribution an die verteilten Speicher durchführt. Dieser Controller benötigt hierfür folgende Informationen:

- Anzahl der Speicher (Anzahl der parallelen Neuro-Chips: k_C)
- Anzahl der Elemente einer Submatrix ($n \cdot k_{PE}$)

Mit diesen beiden Werten werden ein modulo - k_C - Zähler (C1) und ein modulo - ($n \cdot k_{PE}$) - Zähler (C2) vorgeladen. Der C1-Zähler, dessen Wert unmittelbar als Write-Enable-Signal für die k_C verteilten Speicher verwendet werden kann, wird alle $n \cdot k_{PE}$ Takte erhöht. Für je-

den der verteilten Speicher wird ein Register benötigt, in welchem die letzte verwendete Speicheradresse (Offset) abgelegt ist. Durch eine Addition von C2-Zählerstand und Offset-Adresse wird die aktuelle Adresse im W-Speicher ermittelt. Die beschriebene Prozedur läuft solange, bis alle Submatrizen in den W-Speichern abgelegt wurden. Durch die Modulo-Zählweise werden die Speicher zyklisch beschrieben, so daß unabhängig von der Anzahl der parallelen Neuro-Chips die Gewichte stets in der gleichen Reihenfolge bereitgestellt werden können. Basierend auf den erwähnten Zählern und einem Addierer, kann diese automatische Gewichtsdistribution in Hardware implementiert werden. Ein vereinfachtes Schema ist in der folgenden Abbildung 4.27 dargestellt.

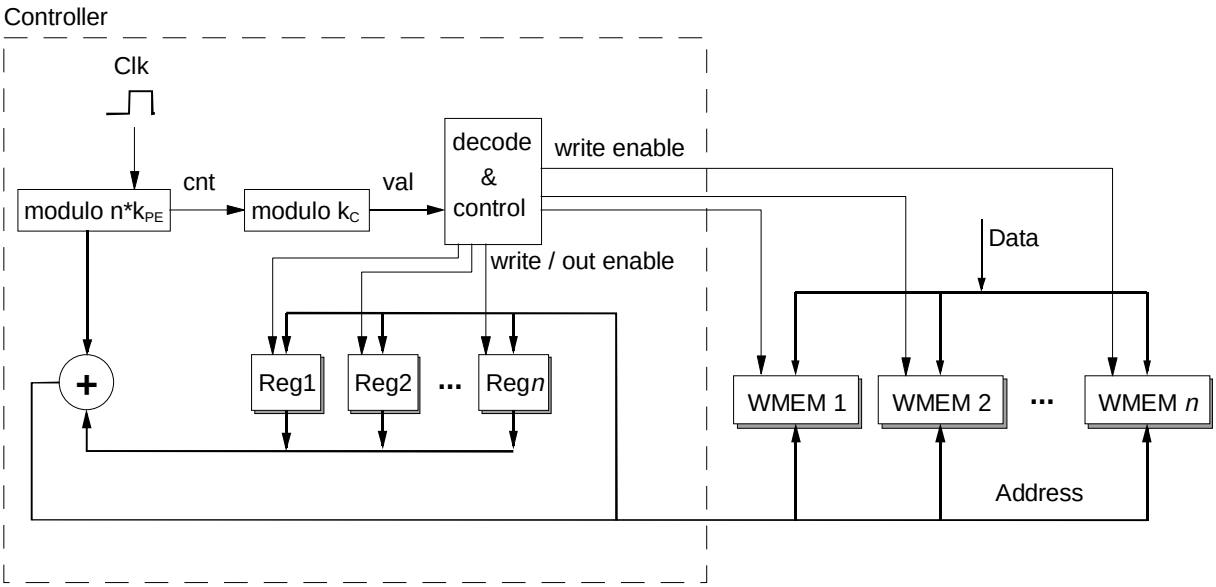


Abbildung 4.27: Vereinfachtes Steuerungsschema der automatischen Gewichts-Distribution

4.7 Zusammenfassung

In diesem vierten Kapitel wurde eine neue Architektur zur Beschleunigung der in der Praxis am häufigsten verwendeten Typen neuronaler Netze (Abbildung 2.1) vorgestellt.

Ausgehend von zwei aus der Literatur bekannten Architekturen zur Beschleunigung neuronaler Operationen (systolische Kette, Broadcast-Architektur), wurde ein neues Kommunikationsschema vorgestellt, das bei unvermindertem Datendurchsatz einen deutlich verringerten Kommunikationsaufwand verursacht. In Kombination mit einer konfigurierbaren ALU ergibt sich eine äußerst effiziente Beschleunigung zahlreicher neuronaler Algorithmen. Hervorzuheben ist, daß diese neue Architektur keine Speicher auf dem Chip erfordert und somit für eine, vor allem bei kleinen Stückzahlen, preisgünstige Implementierung in einem Gate-Array-Prozeß geeignet ist. Der Low-Cost-Gedanke kommt auch bei der Peripherie zum Tragen, da durch das neue Kommunikationsschema die Anzahl externer Bauelemente sehr gering ist.

Wenngleich die ALU dieser neuen Architektur mit Festkommaarithmetik arbeitet, wurde in Zusammenhang mit einer frei ladbaren Look-Up-Tabelle ein Codierungsverfahren vorgestellt, das vor allem beim Training anspruchsvoller Approximationsaufgaben ein verbessertes Konvergenzverhalten erwarten läßt.

Die angestrebte Benutzerfreundlichkeit wird nicht nur durch die Kombination aus neuem Kommunikationsschema und konfigurierbarer ALU, sondern auch durch eine weitgehende Automatisierung der verteilten Speicherung von Gewichten ermöglicht.

Im folgenden Kapitel wird die Implementierung dieser neuen Architektur zur Beschleunigung der Recall-Phase neuronaler Netze auf der Basis einer Sea-Of-Gates-Technologie vorgestellt. Die Konzeption und die Entwicklung einer PC-Einsteckkarte zur Untersuchung und Bewertung dieser neuen Architektur ist ebenfalls Bestandteil des folgenden Kapitels.

5 Implementierung der SAND/1-Architektur

5.1 Einleitung

Unter Berücksichtigung der im vierten Kapitel erarbeiteten theoretischen Grundlagen wurde die Implementierung eines Neuro-Chips auf Basis der IMS-Gate-Forest-0.8µm-Technologie durchgeführt. Das Ziel bei diesem ersten Entwurf ist die Beschleunigung von Operationen der Recall-Phase neuronaler Netze. Das Lernen auf dem Chip wurde hierbei ausgeklammert, ist aber für das Folgeprojekt geplant. Der Neuro-Chip erhielt den Namen „SAND/1“, was als Abkürzung für „**S**imple **A**pplicable **N**eural **D**evice“ steht. Mit dem Zusatz „/1“ soll angedeutet werden, daß es sich hierbei um die erste Version handelt. Zur Beschleunigung PC-basierter Anwendungen wurde eine Einsteckkarte mit PCI-Bus entwickelt, die maximal vier parallel arbeitende SAND/1-Neuro-Chips aufnehmen kann. Eine modular aufgebaute Treiberbibliothek ermöglicht den einfachen Zugriff auf die Hardware und die Integration in Software-Applikationen.

Die folgenden Abschnitte dieses fünften Kapitels geben einen Überblick über den SAND/1-Chip, die PCI-Einsteckkarte und deren Programmierung. Auf der Basis dieser Komponenten wurden zahlreiche Benchmarks durchgeführt, die zur Bewertung der SAND/1-Architektur im sechsten Kapitel verwendet werden.

Der IMS-Gate-Forest-Prozeß

Beim Gate-Forest-Prozeß des IMS Stuttgart handelt es sich um einen CMOS-Sea-Of-Gates-Prozeß mit 0.8 µm effektiver Kanallänge und 2 Metallisierungsschichten. Der Unterschied zwischen einem herkömmlichen (channeled) Gate-Array und einem (free channel) Gate-Array mit Sea-Of-Gates-Struktur besteht in der Anordnung der Basiszellen auf dem Wafer. Beim Channeled-Gate-Array wechseln sich Zellen und Verdrahtungskanäle ab, während bei einer Sea-Of-Gates-Struktur die gesamte Chipfläche mit Zellen bedeckt ist. Bei der Verdrahtung werden einfach Zellen überdeckt, die infolgedessen nicht mehr nutzbar sind. Der Entfall der Verdrahtungskanäle begünstigt aber die automatische Verdrahtung, die eine deutlich kürzere Entwicklungszeit und dadurch geringere Kosten gegenüber einem Channeled-Gate-Array zur Folge hat. Während bei konventionellen Gate-Arrays der Nutzungsgrad annähernd bei 90% liegt, erreicht man bei Sea-Of-Gates-Strukturen jedoch nur zwischen 50% und 75%. Abbildung 5.1 zeigt schematisch den prinzipiellen Aufbau einer 4-Transistor-Basiszelle, wie sie bei beiden Gate-Array-Typen zum Einsatz kommt [Siegl90].

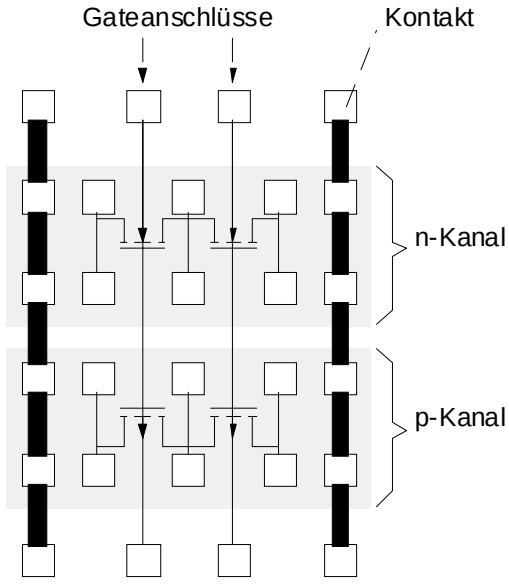


Abbildung 5.1: Aufbau einer 4-Transistor-Basiszelle

Für die Entwicklungswerkzeuge von SYNOPSYS und ViewLogic bietet das IMS ein Designkit an, das die Standardzellbibliothek GFN120 umfaßt [IMS96], welche insgesamt 161 Basis-Core-Zellen und 92 I/O-Zellen zur Verfügung stellt. Der zur Zeit der Entwicklung von

SAND/1 größte angebotene Wafer, der GFN060, hat eine Kapazität von 60.000 NAND2-Äquivalenten, von denen bei einer Ausbeute von 75% 45.000 genutzt werden können. Mittlerweile ist ein weiterer Wafer (GFN120) hinzugekommen, der 120.000 NAND2-Gatteräquivalente besitzt.

5.2 Der Neuro-Chip SAND/1

5.2.1 Architektur des Neuro-Chips (Datenfluß)

Kernstück des Neuro-Chips SAND/1 [Becher97a, Fischer96b, Fischer98] ist eine Anordnung paralleler Prozeßelemente, deren Struktur und Verkettung auf den Ergebnissen des vierten Kapitels aufbauen. Das Schema der Datenpfade ist in der folgenden Abbildung 5.2 dargestellt.

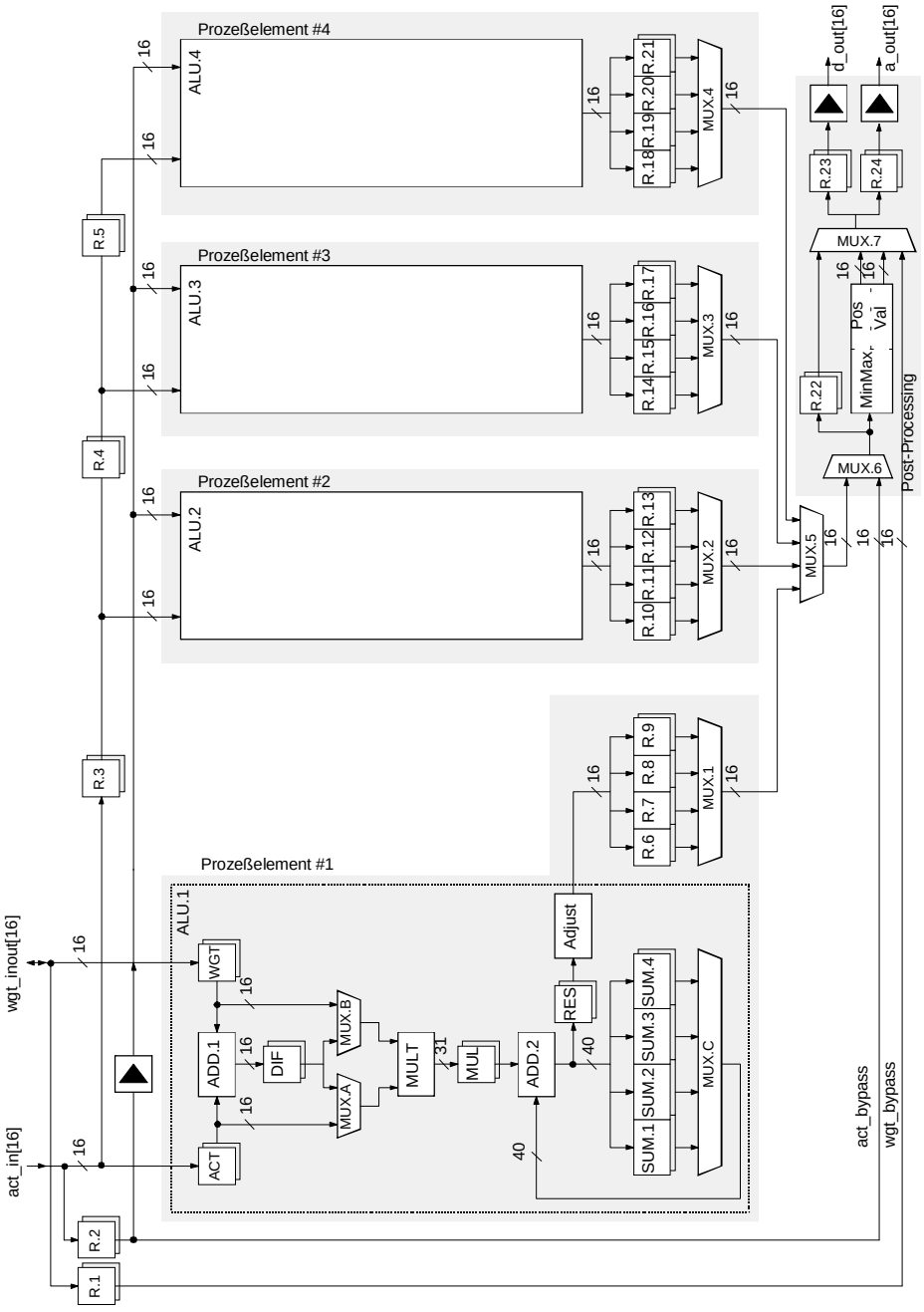


Abbildung 5.2: Architektur des Neuro-Chips SAND/1 (Datenfluß)

Der zum Zeitpunkt der SAND/1-Entwicklung größte zur Verfügung stehende Wafer (GFN060) ließ lediglich die Implementierung von vier parallelen Prozeßelementen zu, von denen jedes ca. 6.200 NAND2-Gatteräquivalente beansprucht.

Neben einer ALU, die im wesentlichen der Abbildung 4.17 entspricht und infolge der vier parallelen Prozeßelemente vier Summationsregister benötigt, wurde eine weitere Registerbank (R.6-R.9, ..., R.18-R.21), bestehend aus vier Registern, in jedem Prozeßelement integriert. Diese Registerbank ermöglicht ein Pipelining auf der Ebene von (Teil-) Schichten eines Netzes, um sicherzustellen, daß während des Auslesens von Ergebnissen eines Zyklus i bereits die Daten des folgenden Zyklus i+1 in die parallelen Prozeßelemente strömen können. Über eine zweistufige Multiplexerstruktur (MUX.1 - MUX.4 / MUX.5) werden die 16 Ergebniswerte Wort für Wort aus dem Chip ausgelesen.

Im Block Postprocessing ist die Extremwertsuche (MinMax) untergebracht, die sowohl auf den Ausgabedaten der Prozeßelemente als auch auf externen Daten (act_bypass) operieren kann. Damit ist sichergestellt, daß auch beim Parallelbetrieb mehrerer SAND/1-Chips, die ihre berechneten Aktivitäten in einem gemeinsamen Speicher verwalten, die Extremwertsuche möglich ist. Neben dem eigentlichen Extremwert (Val) wird für jedes der vier Muster die Adresse (Pos) des Gewinnerneurons ausgegeben.

Um den Schaltungsaufwand auf der Platine beim Laden des Gewichtsspeichers zu verringern, wurde ein weiterer Datenpfad im Chip vorgesehen, der das Laden der Gewichte über den act_in-Bus erlaubt. Zum Test der Gewichtsspeicher wurde der Datenpfad wgt_bypass vorgesehen, der Gewichte an den parallelen Prozeßelementen vorbei, direkt über das Postprocessing-Modul zum Ausgang des Chips leitet.

Die Look-Up-Tabelle, welche außerhalb des Chips in einem 64K-SRAM untergebracht ist, kann über den act_bypass geladen werden. Neben einem Adreßausgang für diese Tabelle besitzt der Chip einen zweiten Ausgang, falls keine Transferfunktion auf die a_j - Werte (Abbildung 4.17) angewendet werden soll. Hierdurch ist gewährleistet, daß ohne ein Neuladen der Tabelle eine lineare und eine nichtlineare Transferfunktion simultan verwendet werden können. Bei RBF-Netzen beispielsweise wird in der verdeckten Schicht eine Gauß'sche Funktion als Aktivierungsfunktion verwendet, während in der Ausgangsschicht lediglich eine lineare Abbildung benötigt wird.

5.2.2 Technische Daten des Neuro-Chips SAND/1

Das DIE des Neuro-Chips SAND/1, dessen Größe ungefähr 12 x 12 mm^2 beträgt, ist in einem PGA-120-Gehäuse untergebracht. Die 120 Pins teilen sich auf 34 Steuerungs- und 64 Datensignale sowie 18 Pins für die Spannungsversorgung (Core / Pad getrennt) und je einen Pin für die Clock und das Reset-Signal auf. Die verbleibenden zwei Pins sind für den Test des Chips (Scan-Pfad) reserviert. Die Abbildung 5.3 zeigt einen Blick auf das DIE des SAND/1-Neuro-Chips.

Der Chip ist in einem 0,8 µm Sea-Of-Gates-Prozeß (IMS Stuttgart) gefertigt und besitzt ca. 44.000 NAND2-Gatteräquivalente. Einige technische Daten das SAND/1 sind in den folgenden Tabellen zusammengefaßt.

Allgemeine Daten

#Addierer	16 / 16 Bit	4
	40 / 31 Bit	4
#Multiplizierer	16 / 16 Bit	4
#Register	16 Bit	34
	31 Bit	4
	40 Bit	20

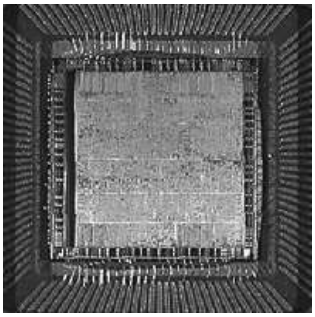


Abbildung 5.3: Blick auf das verdrahtete DIE des SAND/1

Leistungsdaten (Peak-Performance)

Verarbeitungsleistung (50 MHz)	(MCPS)	200
	(MOPS)[12]	600
Transferleistung	act_in	100 MBytes/s
	wgt_inout	100 MBytes/s

Elektrische Eigenschaften

Taktfrequenz	f_{max}	50 MHz	
Versorgungsspannung (min/max)	V_{dd}	4,5V	5,5V
Eingangsspannung	V_i	0V	V_{dd}
Ausgangsspannung	V_o	0V	V_{dd}
Verlustleistung (50 MHz, V_{dd}=5V)	P_{ges}	≈ 4 Watt	

Tabelle 5.1: Technische Daten des SAND/1-Neuro-Chips

5.3 Die SAND/1-PCI-Karte

Zur Beschleunigung PC-basierter Anwendungen und zur Leistungsanalyse des SAND/1-Neuro-Chips wurde eine PC-Einsteckkarte entwickelt [Becher97b, Eppler97, Fischer97]. Ziel bei dieser Entwicklung war es, die hohe Leistung eines oder mehrerer SAND/1-Chips in einem PC bei einfacher Handhabung und möglichst geringen Kosten verfügbar zu machen. Im einzelnen bedeutet dies, daß

- die Anzahl der parallelen SAND/1-Chips variabel sein muß (einfache Skalierbarkeit)
- die Anzahl an zusätzlichen Bauelementen so gering wie möglich sein muß (geringere Kosten)
- das Steuerungskonzept übersichtlich und leicht nachvollziehbar sein muß (einfache Handhabung)

5.3.1 Der Datenfluß auf der SAND/1-PCI-Karte

Die hohe Verarbeitungsleistung des SAND/1-Chips (eine Aktivität bzw. ein Gewicht pro Takt [20 ns]) kann nur dann voll zum Tragen kommen, wenn die ihn umgebende Peripherie die erforderlichen Übertragungsraten unterstützt und die Anbindung an einen Hostrechner mit einer ähnlich hohen Bandbreite möglich ist. Hierfür scheint der sowohl im PC- als auch im

[12] Bei der Berechnung von RBF-Netzen (DISTL2-Operation: 2 Additionen und 1 Multiplikation pro Takt)

Workstation-Bereich etablierte PCI-Bus am besten geeignet zu sein, da er zumindest theoretisch eine maximale Übertragungsrate von 132 MBytes/s bietet. Insbesondere der sogenannte Burst-Transfermode scheint wesentlich zur Leistungsfähigkeit dieses Busses beizutragen, da einer einmal gesendeten Adresse (theoretisch) beliebig viele Datenworte folgen können. Darüber hinaus muß durch die Struktur der Datenpfade auf der PCI-Karte gewährleistet sein, daß sich der Austausch von Daten zwischen der PCI-Karte und dem Hostrechner auf der einen Seite und die Verarbeitung der Daten durch die SAND/1-Chips auf der anderen Seite nicht gegenseitig behindern. Ein dritter Aspekt, der unmittelbaren Einfluß auf die Leistung hat, ist die Organisation der Speicher auf der Karte. Auch bei einem parallelen Betrieb mehrerer SAND/1-Chips muß sichergestellt sein, daß Speicherzugriffe keine Wartezyklen verursachen. Unter Berücksichtigung dieser Punkte wurde die SAND/1-PCI-Karte entwickelt, deren schematischer Aufbau in der Abbildung 5.4 dargestellt ist.

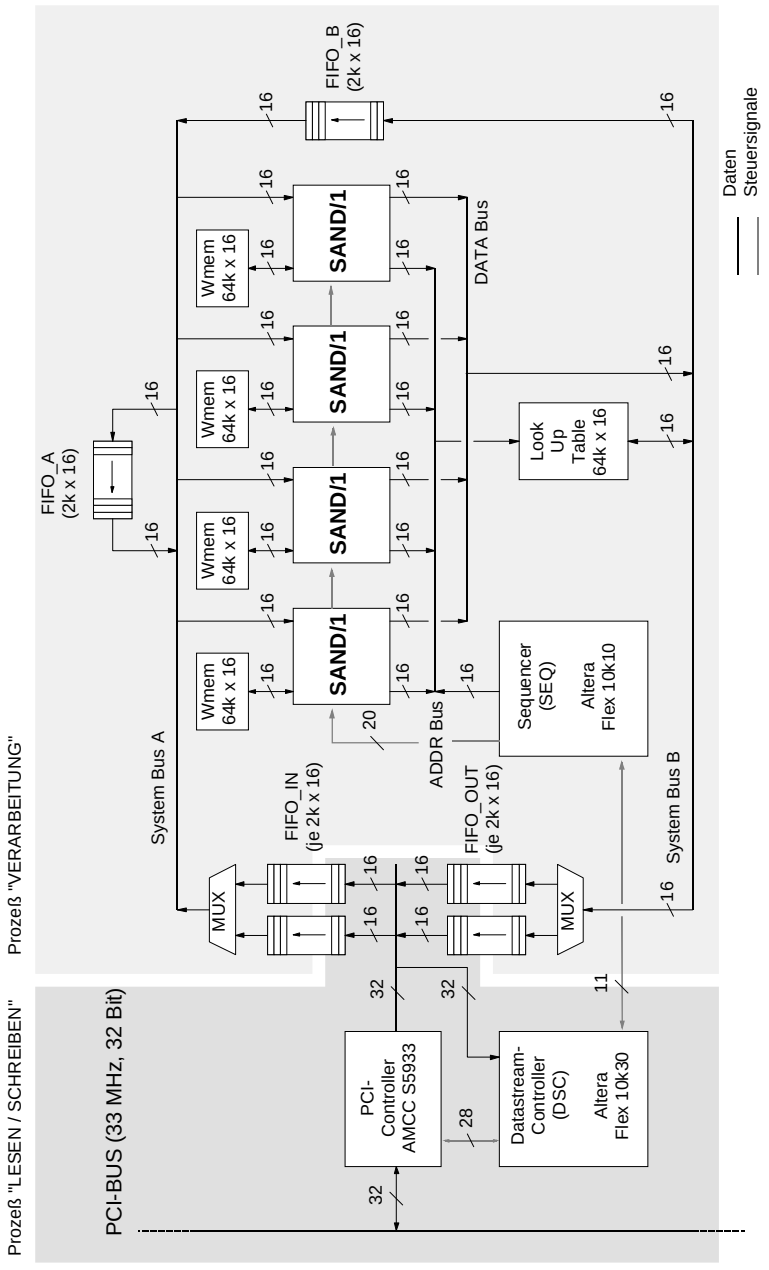


Abbildung 5.4: Blockschaltbild der SAND/1-PCI-Karte

Durch die beiden Speicher FIFO_IN und FIFO_OUT wird eine Entkopplung der beiden Prozesse

- LESEN / SCHREIBEN (Kommunikation mit dem Hostrechner) und
- VERARBEITUNG (der Daten auf SAND/1)

erreicht. Damit können Ergebnisse zum Hostrechner übertragen bzw. neue Eingabedaten gelesen werden, während die SAND/1-Chips auf einem zweiten Datensatz operieren. Anstelle adressierbarer Speicher treten First-In-First-Out-(FIFO)-Speicher, die eine aufwendige Adreßrechnung überflüssig machen. Dies ist möglich, da die Verarbeitungsreihenfolge der Daten unter anderem durch die systolische Verarbeitungsweise der SAND/1-Chips eine zyklische Verarbeitung erlaubt. Für den Fall, daß mehr Neuronen in einer Schicht vorhanden sind als parallele Rechenwerke zur Verfügung stehen, müssen die Eingangsdaten der Schicht mehrfach ausgewertet werden (Partitionierung[13] - siehe Kapitel 4.6.2). Die hierfür erforderliche Zwischenspeicherung erfolgt im rückgekoppelten FIFO_A.

Aufgrund der schichtweisen Verarbeitung der Daten durch SAND/1 ist die erforderliche Anzahl von Arbeitsschritten mindestens so groß wie die Anzahl der Schichten im Netz, d.h. die Aktivitäten verdeckter Schichten müssen zwischengespeichert werden, um anschließend zur Berechnung der nächsten Schicht erneut in den parallelen Rechenwerken der SAND/1-Chips verarbeitet zu werden. Diese Zwischenspeicherung erfolgt im FIFO_B. Handelt es sich bei den berechneten Aktivitäten jedoch um diejenigen der Ausgangsschicht, dann werden diese nicht im FIFO_B, sondern im FIFO_OUT gespeichert. Währenddessen können bereits die nächsten Daten aus dem FIFO_IN gelesen und in den parallelen Rechenwerken verarbeitet werden.

5.3.2 Kontrollfluß und Programmierung

Der Nachteil zahlreicher Parallelrechner zur Beschleunigung neuronaler Netze ist deren oftmals komplizierte Programmierung [lenne95a] bzw. eine uneffektive Auslastung der parallelen Hardware [Wahle94]. Bei der Entwicklung der SAND/1-Architektur standen deshalb neben einer optimalen Nutzung der Ressourcen auch eine einfache Handhabung und Programmierung im Vordergrund. Hierzu trägt zum einen die konfigurierbare (anstelle einer frei programmierbaren) ALU bei. Aber auch die Art der Partitionierung und die Abbildung von Algorithmen in der Art von Gleichung (4.16) auf die SAND/1-Architektur erleichtern die Handhabung.

Der kleinste steuerbare Verarbeitungsschritt ist die Verknüpfung einer (Gewichts-) Submatrix $\underline{W}(4 \times n)$ mit einer Aktivitätsmatrix $\underline{O}(n \times 4)$, welche durch eine Reihe von Attributen beschrieben werden kann. Ausschlaggebend ist die Tatsache, daß jede Netzberechnung in der Art von (4.16) auf diese Suboperation reduziert werden kann, welche stets die gleiche Parameterstruktur besitzt. Werden mehrere SAND/1-Chips auf der Platine parallel betrieben, dann erfolgt eine simultane Verarbeitung solcher Submatrizen, welche jedoch alle zur gleichen Schicht eines Netzes gehören und damit mit einem gemeinsamen Parametersatz

[13] Die Partitionierung aus Sicht des Kontrollflusses wird im folgenden Kapitel 5.3.2 betrachtet.

beschrieben werden können. Aus Sicht des Programmierers muß daher nur die Operation eins SAND/1-Chips parametrisiert werden. Für das Steuerwerk muß darüber hinaus noch bekannt sein, wie viele Submatrizen parallel verarbeitet, d.h. wie viele SAND/1-Chips benötigt werden. Ein Software-Entwicklungswerkzeug für die SAND/1-Hardware muß daher zwei Aufgaben übernehmen:

- Partitionierung einer Gewichtsmatrix in Submatrizen der Größe (4 x n)
- Bestimmen der Parameter zur Konfiguration eines SAND/1-Chips

Die Parameterstruktur setzt sich aus zwei Gruppen zusammen. Die erste Gruppe betrifft alle statischen Parameter, d.h. solche, die für alle Submatrizen einer Schicht gleich sind wie z.B. die Konfiguration der Prozeßelemente (ALU-Funktion, Rundungsmodus), die Größe der Operanden (n), die Extremwertsuche sowie den Typ der Transferfunktion (linear / nichtlinear). Die zweite Gruppe umfaßt alle dynamischen Parameter, die sich von Submatrix zu Submatrix ändern können, wie z.B. die Quelle und das Ziel der Ein- und Ausgangsdaten (FIFO_IN, FIFO_B, FIFO_OUT), die Anzahl der erforderlichen SAND/1-Chips sowie die Position der Submatrix in bezug auf die Gesamtmatrix. Diese Information ist für die Verwaltung der Gewichtsspeicher und die hierfür benötigte Adreßgenerierung erforderlich.

Die folgende Abbildung 5.5 zeigt die gesamte Parameterstruktur.

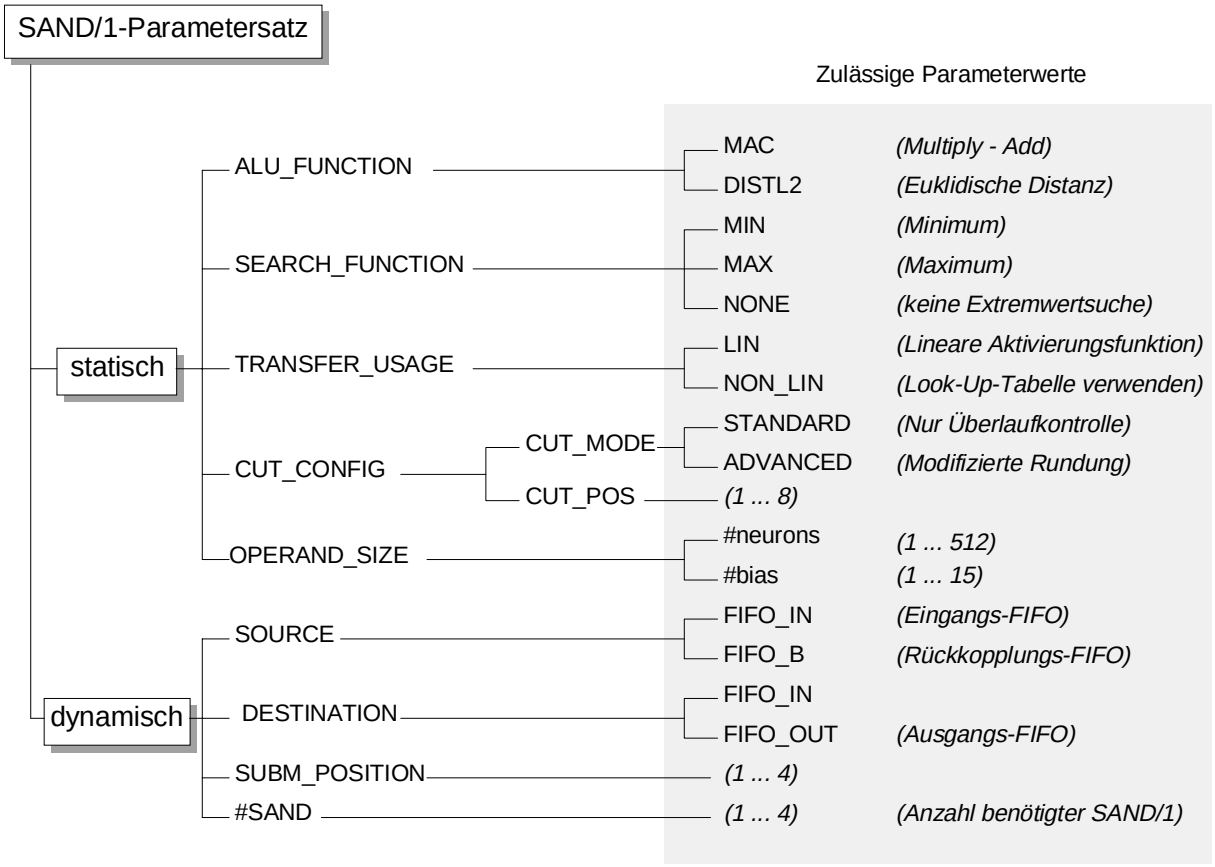


Abbildung 5.5: Parametersatz zur Steuerung der SAND/1-Hardware

Mit diesen 10 Parametern, die sich als ein 32-Bit-Steuerwort darstellen lassen, wird das Verhalten der SAND/1-Chips für $n \cdot 4$ Takte bestimmt. Die Berechnung eines Netzes oder

eines Filters kann damit in eine Reihe solcher Befehlsworte (Macro-Commands) zerlegt werden. Diese Systematik eignet sich besonders für eine automatische Generierung der Steuerworte aus einer Netzbeschreibung. Der Anwender wird von der Problematik der algorithmischen Abbildung eines Netzes auf die parallele Hardware entlastet und kann statt dessen mit einer Beschreibung des Netzes in einer geeigneten Sprache, z.B. CONNECT [Kock96, Fischer97], arbeiten. Anhand des folgenden Beispiels soll dieser Mechanismus verdeutlicht werden.

Beispiel 5.1
Gegeben sei ein Multilayer-Perzeptron mit einer verdeckten Schicht mit 60 Neuronen, einer Eingangsschicht mit 144 Neuronen und einer Ausgangsschicht mit 15 Neuronen. Die SAND/1-PCI-Karte sei mit vier Neuro-Chips bestückt. Die Gewichte und Bias-Werte der verdeckten Schicht und der Ausgangsschicht werden in den beiden Matritzen $\underline{W}_h$ und $\underline{W}_o$

$$\underline{W}_h = \begin{pmatrix} w^h_{1,1} & \cdots & w^h_{1,144} & b^h_1 \\ \vdots & \ddots & \vdots & \vdots \\ w^h_{60,1} & \cdots & w^h_{60,144} & b^h_{60} \end{pmatrix}, \quad \underline{W}_o = \begin{pmatrix} w^o_{1,1} & \cdots & w^o_{1,60} & b^o_1 \\ \vdots & \ddots & \vdots & \vdots \\ w^o_{15,1} & \cdots & w^o_{15,60} & b^o_{15} \end{pmatrix}$$

zusammengefaßt. Mit vier parallelen SAND/1-Chips stehen 16 parallele Prozeßelemente zur Verfügung, so daß die verdeckte Schicht in

$$S_h = \left\lceil \frac{60}{16} \right\rceil = 4$$

Schritten berechnet wird. Während bei den ersten drei Schritten alle vier SAND/1-Chips benötigt werden, genügen beim vierten Schritt drei Chips, um die verbleibenden 12 Neuronen zu berechnen. Für die vier Schritte zur Berechnung der verdeckten Schicht werden vier Macro-Commands benötigt, deren Parameter unmittelbar aus der Partitionierung abgeleitet werden können (Tabelle 5.2).

Parameter	**Macrocommand1**	**Macrocommand2**	**Macrocommand3**	**Macrocommand4**
[statisch]				
ALU_FUNCTION	MAC			
SEARCH_FUNCTION	NONE			
TRANSFER_USAGE	NON_LIN			
CUT_MODE	ADVANCED			
CUT_POS	(datenabhängig)			
OPERAND_SIZE	#neurons=144, #bias=1			
[dynamisch]				
SOURCE	FIFO_IN	FIFO_IN	FIFO_IN	FIFO_IN
DESTINATION	FIFO_B	FIFO_B	FIFO_B	FIFO_B
SUBM_POSITION	1 (=Anfang)	0 (=Mitte)	0 (=Mitte)	2 (=Ende)
#SAND	4	4	4	3

Tabelle 5.2: Macro-Commands zur Berechnung der verdeckten Schicht (Beispiel 5.1)

Die 15 Neuronen der Ausgangsschicht können in einem Schritt berechnet werden, wobei das vierte Prozeßelement des vierten SAND/1-Chips ungenutzt bleibt. Dies spiegelt sich nicht im Macro-Command selbst wider, muß aber bei der Aufbereitung der Gewichte berücksichtigt werden, d.h. die vierte Zeile der Gewichtsmatrix $\underline{W_h}$ wird mit Nullen aufgefüllt. Der Macro-Command zur Berechnung der Ausgangsschicht ergibt sich aus Tabelle 5.3:

Parameter	**Macrocommand5**
[statisch]	
ALU_FUNCTION	MAC
SEARCH_FUNCTION	NONE
TRANSFER_USAGE	NON_LIN
CUT_MODE	ADVANCED
CUT_POS	(datenabhängig)
OPERAND_SIZE	#neurons=60, #bias=1
[dynamisch]	
SOURCE	FIFO_B
DESTINATION	FIFO_OUT
SUBM_POSITION	3 (=Anfang&Ende)
#SAND	4

Tabelle 5.3: Macro-Commands zur Berechnung der Ausgangsschicht (Beispiel 5.1)

■

Mit Hilfe eines einfachen Algorithmus, welcher Bestandteil einer automatischen Generierung ist, läßt sich der dynamische Parameter *#SAND* bestimmen. Die PCI-Karte sei mit *#MAXS* Neuro-Chips bestückt, die Anzahl der zu berechnenden Neuronen in der Schicht sei *m* und der gesuchte Parameter sei *#SAND*.

Algorithmus 5.1

für alle l Schichten

Gesamtzahl der Schritte $S = \left\lceil \frac{m}{\#MAXS \cdot 4} \right\rceil$

für alle Schritte von s=0 bis s=S-1

$$\#SAND[s] = \#MAXS - \frac{4 \cdot \#MAXS - \left(4 \cdot \#MAXS \bmod (m - s \cdot 4 \cdot \#MAXS)\right)}{4}$$

Ende Schleife s

Ende Schleife Schichten

■

Sowohl beim MA16 [Siemens93] als auch beim CNAPS 10xx [Salmen95] muß der Anwender auf der Basis einzelner Matrixoperationen die Verarbeitungsschritte festlegen und in einer geeigneten Form codieren. Beim MA16 kommt die Schwierigkeit hinzu, daß hier nur Submatrizen der Größe (4 x 4) verarbeitet werden, was in vielen Fällen eine zusätzliche Weiterverarbeitung (Matrixaddition) der Zwischenergebnisse zur Folge hat.

Die Separierung der Verarbeitung in mehrere Suboperationen der oben beschriebenen Art spiegelt sich auch in der Steuerungsstruktur wider. Die parallel angeordneten SAND/1-Chips auf der PCI-Karte bilden ein SIMD-Prozessorfeld, das infolgedessen zur Verarbeitung von Submatrizen der Größe (4 x *n*) nur ein zentrales Steuerwerk benötigt. Dieses Steuerwerk führt auf der Basis der erläuterten 32-Bit-Befehlswörter die Konfiguration sämtlicher Prozeßelemente durch, erzeugt die Adressen für die Gewichtsspeicher und speichert die Ausgangsaktivitäten in einem der beiden möglichen FIFOs (FIFO_B, FIFO_OUT) ab. Dieses Steuerwerk, im folgenden Sequencer (SEQ) genannt, ist in einem Altera Flex 10K10 FPGA implementiert und wird wie die SAND/1-Chips mit 50 MHz getaktet.

Zur Verwaltung aller Suboperationen eines Netzes bzw. einer Schicht wird eine weitere Steuerungskomponente benötigt, die die Schnittstelle zwischen dem PCI-Controller und dem Sequencer bildet und den Datenstrom zwischen Hostrechner und der SAND/1-PCI-Karte koordiniert. Diese Komponente, im folgenden Data-Stream-Controller (DSC) genannt, wurde ebenfalls in einem Altera-FPGA der 10K-Serie implementiert und wird mit zwei verschiedenen Takten (33MHz - PCI, 50MHz - SAND/1) versorgt. Der DSC besitzt einen internen Programmspeicher für sämtliche Macro-Commands, die für die Verarbeitung eines Netzes bzw. Filters benötigt werden. Diese auf dem Hostrechner generierten Befehlsworte werden mittels eines Befehls (**`load_net_cfg)`** in den internen Speicher des DSC geschrieben und nach Eintreffen des Startbefehls (**`load_and_calc`**) Byte für Byte ausgelesen und an den SEQ weitergeleitet. Nach Abarbeitung aller Macro-Commands erzeugt der DSC einen Interrupt, welcher den Hostrechner zum Lesen der Ergebnisse veranlaßt.

Neben den 32-Bit-Macro-Commands, welche vom Sequencer interpretiert werden und der Verarbeitung einzelner (4 x *n*) Suboperationen dienen, wurde ein weiterer Befehlssatz zur globalen Steuerung des Datenflusses zwischen einem Hostrechner und der SAND/1-PCI-Karte eingeführt. Hierzu zählen die bereits erwähnten Befehle zum Laden von Macro-Commands in den internen Speicher des DSC und das Starten einer Operation sowie Befehle zum Laden der Gewichtsspeicher und der Look-Up-Tabelle. Dieser globale Befehlssatz wird vom DSC interpretiert, welcher mit Hilfe eines einfachen Handshake-Protokolls die Kommunikation mit dem Hostrechner durchführt.

Die folgende Abbildung 5.6 zeigt anhand eines stark vereinfachten Schemas das Zusammenwirken der beiden Controller (DSC, SEQ) und deren Kopplung an einen Hostrechner auf der einen Seite und das SAND/1-Prozessorfeld auf der anderen Seite sowie die Struktur einer Host-Applikation, die auf dem Software-Treiber aufbaut.

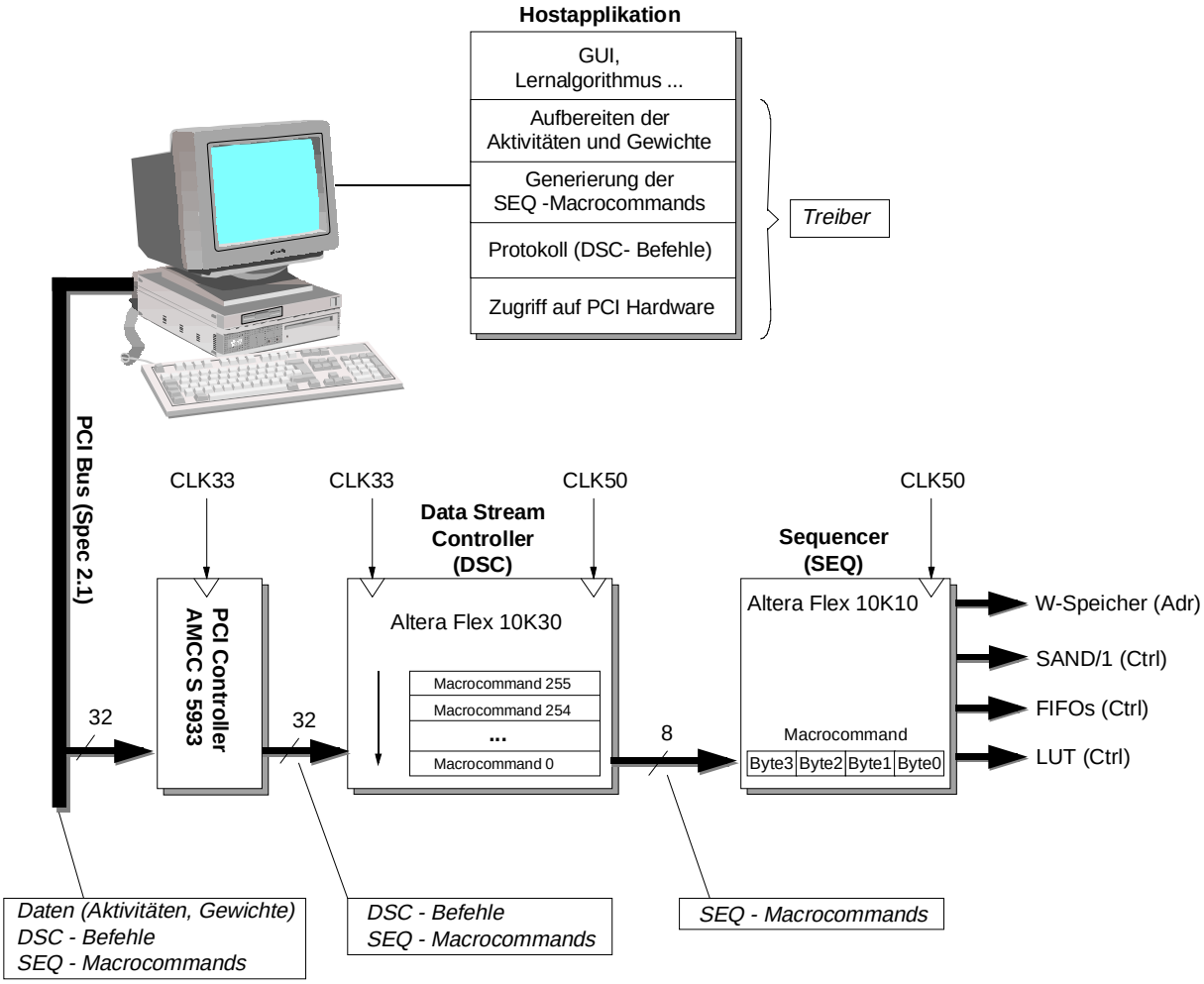


Abbildung 5.6: Schematische Darstellung der Kommunikation und Steuerung des SAND/1-PCI-Boards

Auf der Basis dieser hierarchischen Steuerung (Partitionierung der Gewichtsmatrix, Generierung der Macro-Commands, Kommunikation mit dem DSC) wurde eine Treiber-Software entwickelt, bei der lediglich die unterste Ebene plattformspezifische Komponenten in Form von Zugriffen auf die PCI-Hardware enthält, während alle darüberliegenden Teile in der Form von C/C++-Programmen leicht auf andere Plattformen[14] portiert werden können.

Dem Anwender werden Schnittstellen auf zwei unterschiedlichen Stufen zur Verfügung gestellt - für die gebräuchlichsten Typen neuronaler Netze[15] stehen C-Funktionen [dataf98] zum Erzeugen und Laden der Netzinformation, der Gewichte, Aktivitäten und Einträge für die Look-Up-Tabelle zur Verfügung. Sie ermöglichen eine intuitive Nutzung der neuronalen Hardware ähnlich einem Software-Simulationswerkzeug. Zur Generierung der Netzinformation für ein Multilayer-Perzeptron mit einer verdeckten Schicht wird einmalig die Funktion

[14] Die Treiberbibliothek wurde für Intel x86 basierte Systeme auf MS-DOS 6.2, Windows 3.11, Windows 95, Windows NT 4.0 und LINUX portiert.
[15] Siehe Kapitel 2.2

```
SAND_ERROR SAND_LoadMLP1(  SAND_LAYER_SIZE sz_in,
                           SAND_LAYER_SIZE int sz_h,
                           SAND_LAYER_SIZE int sz_out,
                           SAND_TRANSFER_MODE act_h
                           SAND_TRANSFER_MODE act_out,
                           SAND_WEIGHTS_TABLE ws_h
                           SAND_WEIGHTS_TABLE ws_out,
                           SAND_BIAS_TABLE bs_h,
                           SAND_BIAS_TABLE bs_out)
```

aufgerufen, um die SAND/1-PCI Karte für dieses Netz zu konfigurieren. Danach werden mittels

```
SAND_ERROR SAND_ApplyMLP(  unsigned int n,
                           SAND_ACTIVITIES_INPUT x,
                           SAND_ACTIVITIES_OUTPUT y)
```

n Muster auf der SAND-Hardware berechnet und an den Aufrufer zurückgegeben. In gleicher Weise werden RBF- und Kohonen-Netze gehandhabt. Eine zweite, darunterliegende Schnittstelle, erlaubt die Erweiterung der High-Level-Bibliothek um anwenderspezifische Funktionen. Hierfür werden Routinen zum Laden der Gewichte, zur Generierung und zum Laden der Macro-Commands und zur Aufbereitung der Aktivitäten zur Verfügung gestellt [Becher97a]. Auch auf dieser Ebene der Software sind für den Anwendungsentwickler keine detaillierten Kenntnisse der parallelen SAND/1-Architektur erforderlich.

Die Treiberbibliothek wurde mittlerweile in Form einer DLL[16] in ein Softwaresystem zur Simulation neuronaler Netze eingebunden [Becher97b]. Diese Software besitzt eine leicht zu bedienende Oberfläche (Windows95/98) und erlaubt den Entwurf neuronaler Netz unter Benutzung grafischer Elemente und einer speziellen Entwurfssprache (Connect) [Kock96]. Mit Hilfe des Softwaresimulators können trainierte Netze unmittelbar auf das PCI-Board geladen und dort beschleunigt ausgeführt werden.

5.3.3 Design und Test

Bei der Entwicklung der gesamten SAND/1-Hardware (SAND/1-Chip und PCI-Board) wurde von Beginn an Wert auf eine kurze Entwicklungszeit gelegt. Der Alptraum jedes Entwicklers ist ein Redesign, das die Kosten und die Dauer einer Entwicklung in die Höhe treibt. Deshalb wurde schon zu Beginn der Entwicklung eine Simulation auf Board-Ebene vorgesehen, um die besonders fehleranfälligen Schnittstellen zwischen Steuerungs- und Verarbeitungseinheit (Controller - SAND/1) und Software - Hardware bereits während der Entwicklungsphase ausgiebig testen zu können. Da die Entwicklung der Controller ohnehin eine Modellierung auf Verhaltensebene erforderte, wurde als gemeinsame Basis die Hardware-Beschreibungssprache VHDL (IEEE-1076-87) gewählt.

[16] DLL bedeutet **D**ynamic **L**ink **L**ibrary

Designflow bei der Entwicklung des Neuro-Chips SAND/1
Auch für den Neuro-Chip SAND/1 wurde zuerst eine Verhaltensbeschreibung in VHDL durchgeführt, auf deren Basis verschiedene Varianten der Architektur untersucht werden konnten. Bis auf die parallelen ALUs wurde hierbei im Prinzip schon eine synthesefähige RT-Modellierung realisiert. Im Rahmen dieser Verhaltensbeschreibung wurde eine Testbench entwickelt, die es ermöglicht, Szenarien zu simulieren, wie sie später beim Zusammenwirken eines SAND/1-Chips mit einem Controller (Sequencer) realistisch sind. Zu diesem Zweck wurden die Ausgangssignale des Sequencers für verschiedene Steuerungsabläufe in einer Trace-Datei abgelegt, um sie später für die Stand-Alone-Simulation von SAND/1 verwenden zu können.

Für die Synthese wurde die Verhaltensbeschreibung der ALU durch eine separat erstellte Netzliste ersetzt. Diese Netzliste besteht aus den Registern der ALU (Abbildung 4.17) sowie den Rechenwerken (Multiplizierer, Addierer), die mit einem Modulgenerator [Marwed92] des IMS erstellt wurden. Mit Hilfe der Testbench wurde das Ergebnis der Synthese überprüft. Die Ermittlung des kritischen Pfades ergab, daß dieser erwartungsgemäß zwischen den Registern MUL und SUM.*e* der ALU liegt und eine Verzögerung von 16 ns verursacht (worst-case-Simulation).

Testbarkeit des SAND/1-Designs
Jeder Chip wird im allgemeinen nach der Herstellung und vor dem Verpacken einem ersten Test unterzogen, um Produktionsfehler zu erkennen und somit fehlerhafte DIEs sofort auszusortieren. Beim SAND/1 werden im wesentlichen zwei unterschiedliche Tests durchgeführt:

- Test mittels Scan-Pfad
- Test durch Anlegen von Testvektoren und Vergleich mit einer Sollausgabe

Für den erstgenannten Test wurde bereits bei der Synthese ein sogenannter Scan-Pfad [Walds96] eingefügt. Die zum Test erforderlichen Vektoren wurden mit einem Tool des Synthesewerkzeugs SYNOPSYS erzeugt. Der zweite Test basiert auf Vektoren, die mit Hilfe der bereits erläuterten Testbench erzeugt wurden. Neben den sonst üblichen Kriterien für die Erzeugung von Testvektoren (hohe Fehlerabdeckung) wurde besonderer Wert darauf gelegt, einige „echte“ Szenarien zu simulieren. Dies betrifft in erster Linie längere Akkumulationszyklen der ALU. Da diese Vektoren den Einsatz unter realen Bedingungen widerspiegeln, konnten sie für einen Speed-Test (50 MHz, verpackter Chip) verwendet werden, um Stromaufnahme und Temperaturstabilität zu untersuchen.

Testbarkeit der PCI-Karte
Bei einer komplexen Schaltung wie der SAND/1-PCI-Karte stellt die Inbetriebnahme einen zeitraubenden Schritt dar. Zahlreiche Fehler, die sich oftmals nur in einem einzigen Symptom äußern, können gleichzeitig auftreten. Um diese Arbeit zu vereinfachen, wurden zahlreiche Testmöglichkeiten auf dem PCI-Board vorgesehen. In einem ersten Schritt kann die Kommunikation mit dem PCI-Controller analysiert werden. Hierzu sendet der Host-Rechner Konfigurationsdaten über den PCI-Bus an den Controller, der diese in einem separaten EEPROM abspeichert und nach Aufforderung wieder an den Host zurücksendet.

Danach kann damit begonnen werden, die Speicher und die Verarbeitungseinheiten auf der Platine sukzessive zu testen. Zuerst wird der FIFO_OUT untersucht. Hierzu werden von einem der Controller bekannte Testmuster erzeugt, im FIFO_OUT abgelegt, vom Hostrechner gelesen und schließlich ausgewertet. Die folgenden Schritte umfassen den Test des FIFO_IN, des FIFO_A und schließlich des FIFO_B. Danach werden die Look-Up-Tabelle und die Gewichtsspeicher mit Zufallszahlen gefüllt und anschließend wieder gelesen. Zuletzt wird eine vollständige Netzkonfiguration an das SAND/1-PCI-Board gesendet, um damit die SAND/1-Chips zu testen.

Diese Schritte sind in die Initialisierungsroutinen der Treiberbibliothek integriert und werden automatisch ausgeführt. Zusätzlich wurde ein Testprogramm entwickelt, das die Modifikation weiterer Parameter (Blockgrößen, Anzahl der Testzyklen etc.) erlaubt.

5.4 Zusammenfassung

Auf der Basis einer Sea-Of-Gates-Technologie wurde die im vierten Kapitel vorgestellte Architektur zur Beschleunigung von Operationen der Recall-Phase neuronaler Netze in Hardware implementiert, gefertigt und erfolgreich getestet. Eine ebenfalls im Rahmen dieser Arbeit konzipierte und entwickelte PCI-Karte erlaubt den parallelen Betrieb von bis zu vier SAND/1-Chips. Die einfache Handhabung des gesamten Systems beruht sowohl auf der neuen Architektur als auch auf einem neuen Steuerungskonzept, das eine einfache Parametrisierung der Hardware erlaubt.

Mit einem Personalaufwand von lediglich 1,5 Mannjahren wurde der SAND/1-Neuro-Chip konzipiert und entwickelt. In einem ähnlichen Rahmen bewegte sich der Aufwand für die Entwicklung des PCI-Boards. Vor allem in der Entwicklungszeit von SAND/1 spiegelt sich der Vorteil des Gate-Array-Designs wider (geringe Entwicklungskosten). Die Herstellungskosten von ca. 200.- DM bei Stückzahlen von weniger als 500 Chips verdeutlichen, daß sich die neue Architektur als echte Low-Cost-Lösung in Hardware implementieren läßt.

Zur Beurteilung der Leistungsfähigkeit wurden zahlreiche Benchmarks durchgeführt, die Gegenstand des folgenden Kapitels sind. Darüber hinaus soll das nächste Kapitel klären, in wieweit sich moderne Standardprozessoren vom Neuro-Chip-SAND/1 in puncto Leistung unterscheiden.

6 Ergebnisse und Bewertung

In diesem sechsten Kapitel wird die Leistungsfähigkeit der SAND/1-Architektur unter verschiedenen Gesichtspunkten analysiert und bewertet [Gemmeke97]. Besondere Bedeutung kommt dem Vergleich mit modernen Prozessoren hoher Leistung bei, da der im Rahmen dieser Arbeit entstandene Neuro-Chip-SAND/1 mit dem Ziel entwickelte wurde, Operationen neuronaler Netze wesentlich schneller als Standardprozessoren auszuführen. Ein Vergleich der verschiedenen Architekturen am Ende dieses Kapitels verdeutlicht, welches Leistungspotential die jeweiligen Prozessoren besitzen, welcher Teil davon wirklich genutzt wird und welche Optimierungen denkbar wären.

6.1 Leistungsanalyse der SAND/1 Architektur

Bereits in Kapitel 2.4.2 dieser Arbeit wurde erläutert, daß aufgrund der Vielfalt der Implementierungsmöglichkeiten neuronaler Operationen eine objektive Leistungsbeurteilung nur eingeschränkt möglich ist. Hinzu kommen weitere Aspekte wie z.B. die Fragestellung, unter welchen Bedingungen die Leistung zu ermitteln ist. Eine möglichst objektive Analyse sollte daher verschiedene Gesichtspunkte berücksichtigen. Die folgenden Betrachtungen werden sich daher zunächst auf die Leistung der PCI-Karte als Stand-Alone-System konzentrieren. Diese Untersuchungen berücksichtigen lediglich den Datenfluß auf der PCI-Karte und besitzen somit auch für andere Applikationen mit SAND/1 und seiner erforderlichen Peripherie (Speicher, Controller) Gültigkeit. Darüber hinaus erlauben diese Ergebnisse einen objektiven Vergleich mit anderen Prozessoren. Der zweite Teil der Betrachtungen schließt dann die Kopplung der SAND/1-PCI-Karte mit einem Hostrechner über den PCI-Bus ein. Das Ziel dieser Untersuchungen ist, konkret zu ermitteln, welche Beschleunigung einer PC-basierten Anwendung unter Einsatz der SAND/1-PCI-Karte möglich ist.

6.1.1 Verarbeitungsleistung der SAND/1-PCI-Karte

Aus der Simulation des SAND/1 Neuro-Chips, des Data-Stream-Controllers, des Sequencers und der beteiligten Speicher können zwei einfache Algorithmen zur Bestimmung der Leistung bei einer vorgegebenen Netztopologie abgeleitet werden. Dies ist möglich, da einerseits sämtliche Komponenten der SAND/1-PCI-Karte, mit Ausnahme der Kopplung an den PCI-Bus, synchron zu einer Clock (50MHz) arbeiten, und andererseits keine Interrupts oder Sprungverzweigungen existieren. Damit ist die Dauer einer Berechnung genau zu bestimmen und hieraus die Leistung in MCPS abzuleiten.

Die Gültigkeit der Algorithmen wurde anhand zahlreicher Beispiele mit einem Logic-Analyzer auf der Platine überprüft. Im folgenden sind die beiden Algorithmen angegeben, die für alle weiteren Betrachtungen herangezogen werden.

Algorithmus 6.1: Leistungsermittlung p_{MCPS} in MCPS für MLP- und RBF-Netze

Anzahl Takte $t_{ges}=0$
für alle l *Schichten (Index* ν*)*

$$\textit{Gesamtzahl der Schritte } S[\nu] = \left\lceil \frac{m[\nu]}{\#MAXS \cdot 4} \right\rceil$$

wenn $\nu \leq (l\text{-}1)$*, dann*
$t_{ges}=t_{ges}+S[\nu]\cdot(n[\nu]\cdot 4+20)$
sonst
$t_{ges}=t_{ges}+S[\nu]\cdot(n[\nu]\cdot 4+120)$
Ende Schleife ν

$$p_{MCPS} = \frac{\#Gewichte\ im\ Netz \cdot 4}{t_{ges} \cdot 20ns} \cdot 10^{-6}$$

■

Erläuterungen zum Algorithmus 6.1

In einer Schleife wird für alle l Schichten (Index ν) des Netzes (l*-1* verdeckte Schichten und eine Ausgangsschicht) die Anzahl benötigter Takte ermittelt. Jede Schicht hat $n[\nu]$ Neuronen in der vorhergehenden Schicht und $m[\nu]$ Neuronen in der zu berechnenden Schicht. Ausschlaggebend ist die Anzahl der Schritte $S[\nu]$, die besagt, wie oft die Aktivitäten der Sendeschicht $n[\nu]$ den parallelen SAND/1-Chips präsentiert werden müssen, um mit $\#MAXS \cdot 4$ parallelen Rechenwerken ($\#MAXS$: Anzahl paralleler SAND/1-Chips) $m[\nu]$ Neuronen zu berechnen. Da SAND/1 stets vier Muster überlappend verarbeitet, wird die Anzahl der gelesenen Aktivitäten noch mit vier multipliziert. Bedingt durch Latenzzeiten der Controller vergehen nach jedem Schritt 20 Takte in verdeckten Schichten bzw. 120 Takte in der Ausgangsschicht, bis die $n[\nu]$ Eingangsaktivitäten den parallelen Rechenwerken der SAND/1-Chips erneut präsentiert werden.

Algorithmus 6.2: Leistungsermittlung p_{MCPS} in MCPS für Kohonen-Netze

Anzahl Takte $t_{ges}=0$

$$\textit{Gesamtzahl der Schritte } S = \left\lceil \frac{m}{\#MAXS \cdot 4} \right\rceil$$

$t_{ges}=t_{ges}+S\cdot(n\cdot 4+20)$

$t_{ges}=t_{ges}+m\cdot 4+20$
$t_{ges}=t_{ges}+4+20$

$$p_{MCPS} = \frac{\#Gewichte\ im\ Netz \cdot 4}{t_{ges} \cdot 20ns} \cdot 10^{-6}$$

■

Erläuterungen zum Algorithmus 6.2

Im Gegensatz zu Algorithmus 6.1 wird bei Kohonen-Netzen nur eine Schicht berechnet, so daß die Schleife entfällt. Die Berechnung der Anzahl benötigter Schritte ist äquivalent zur Berechnung in Algorithmus 6.1. Nach Abschluß der parallelen Berechnung werden zunächst alle Daten im FIFO_B zwischengespeichert. Aus diesen Daten wird in einem weiteren Schritt getrennt für jedes Muster das Minimum ermittelt. Die hierfür erforderliche Anzahl von Takten hängt von der Anzahl der Neuronen in der betrachteten Schicht (m) und einer konstanten Latenzzeit der Controller ab. Nach vier Takten Latenzzeit[17] im SAND/1-Neuro-Chip und weiteren 20 Takten Latenzzeit durch die Controller stehen die Ergebnisse (Minima und deren Position in der Kohonen-Schicht) im Ausgangs-FIFO bereit.

Verifikation der Algorithmen

Die Verifikation der beiden Algorithmen wurde sowohl durch Messungen auf der Platine als auch mit Hilfe eines Testprogramms durchgeführt. Mit Hilfe eines Logic-Analyzers wurde gemäß dem Meßaufbau in Abbildung 6.1 die Zeitdifferenz $\Delta\tau$ zwischen dem Lesesignal des Eingangs-FIFOs und dem Schreibsignal des Ausgangs-FIFOs ermittelt, welche dem Produkt aus t_{ges} (Algorithmen 6.1 und 6.2) und der Zykluszeit (20ns) entsprechen sollte. Dieser Zusammenhang konnte durch die Messungen bestätigt werden.

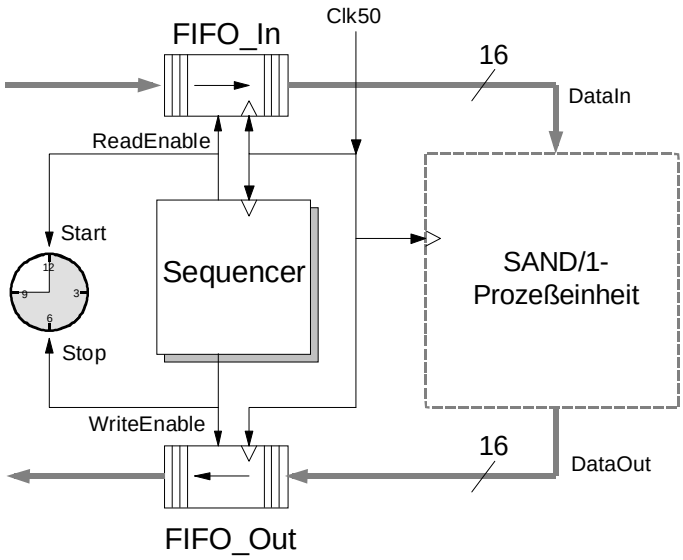


Abbildung 6.1: Meßaufbau zur Verifikation des Timing-Modells

Die bestehenden Treiberfunktionen wurden so erweitert, daß die Zeit zwischen dem Übertragen der Daten zur SAND/1-PCI-Karte hin und anschließend zum Hostrechner zurück gemessen werden kann. Da nur eine sekundengenaue Auflösung möglich ist, werden einige hunderttausend Zyklen in der oben beschriebenen Art innerhalb einer Schleife durchgeführt, und es wird deren Gesamtzeit ermittelt (Abbildung 6.2). Um negative Einflüsse durch Swapping oder Task-Wechsel zu vermeiden, wurden die Benchmarks unter dem Betriebsystem MS-DOS durchgeführt.

[17] die Latenzzeit von vier Takten entspricht der Länge der Pipeline im SAND/1 zur Bestimmung des Minimums.

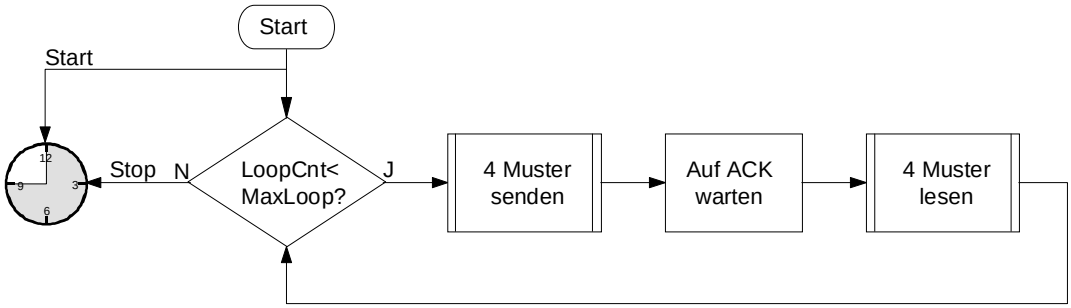


Abbildung 6.2: Ermittlung der Zeit für die Verarbeitung von vier Mustern in einer Schleife

Insbesondere die Größenordnung der Latenzzeiten läßt sich mit Hilfe der Software überprüfen. Zuerst wurde ein MLP1-Netz (512-32-1), danach ein MLP1-Netz (512-33-1) auf der SAND/1-PCI-Karte (vier Neuro-Chips) 3.000.000 mal berechnet. Für beide Netze ist die Übertragungszeit auf dem PCI-Bus identisch. Durch das zusätzliche Neuron in der verdeckten Schicht müssen nochmals alle 512 Eingangswerte gelesen werden, da sich die Anzahl der Schritte beim 512-33-1-Netz auf S=3 gegenüber dem 512-32-1-Netz (S=2) erhöht hat. In der Ausgangsschicht dauert die Verarbeitung ebenfalls länger, allerdings kommt hier nur ein weiterer Akkumulationsschritt hinzu. Da SAND/1 vier Muster überlappend verarbeitet, ergibt sich insgesamt eine Erhöhung der Verarbeitungszeit (ohne Beachtung der Latenzzeit durch die Controller) von

$$\Delta\tau_{calc} = (512+1)\cdot 4\cdot 20ns = 41{,}04\mu s$$

Die mit Hilfe der Software gemessenen Zeiten, die sowohl Übertragungszeiten des PCI-Busses als auch sämtliche Latenzzeiten enthalten, sind in der folgenden Tabelle aufgeführt:

Netzgröße	Dauer für 3.000.000 Zyklen	Dauer für einen Zyklus
512-32-1	645 s	215,00 µs
512-33-1	769 s	256,33 µs

Tabelle 6.1: Gemessene Zeiten zur Ermittlung der Controller-Latenzzeiten

Da die Übertragungszeiten für beide Netze identisch sind, beinhaltet die Differenz von

$$\Delta\tau_{gesamt} = 41{,}33\mu s$$

lediglich noch die Latenzzeit für einen Verarbeitungsschritt (512 · 4 Aktivitäten), die sich somit zu

$$\Delta\tau_{Latenz} = \Delta\tau_{gesamt} - \Delta\tau_{calc} = 293{,}33\,ns$$

ergibt, was bei 50 MHz Taktfrequenz ungefähr 15 Takten entspricht.

Die Differenz zu den im Modell (Algorithmus 6.1) angenommen 20 Takten rührt von der relativ ungenauen Zeitmessung und anschließender Differenzbildung zweier fehlerbehafteter Zeiten her. In gleicher Weise läßt sich mit Hilfe zweier MLP0-Netze, die sich in der Ausgangsschicht um ein Neuron unterscheiden, die Größenordnung der Latenzzeit für die Ausgangsschichten überprüfen.

6.1.2 Tatsächlich nutzbare Leistung

Im zweiten Teil der Leistungsanalyse rückt nun die auf einem PC tatsächlich nutzbare Leistung in den Mittelpunkt. Für den Anwender ist vor allem interessant, welche Beschleunigung durch den Einsatz der SAND/1-PCI-Karte zu erreichen ist. Um diese Fragestellung genauer zu untersuchen, wurden auf der Basis der Treibersoftware unter MS-DOS Benchmarks durchgeführt, deren Ergebnisse im folgenden Abschnitt dargestellt sind.

Da Leistungsverluste in erster Linie auf die Übertragung der Daten über den PCI-Bus zurückzuführen sind, werden im ersten Versuch MLP1-Netze betrachtet, die sich lediglich in der Größe ihrer verdeckten Schicht unterscheiden. Betrachtet wird die Verarbeitung von Multilayer-Perzeptron-Netzen mit 512 Neuronen in der Eingangsschicht und 16 Neuronen in der Ausgangsschicht. Die Anzahl der Neuronen in der verdeckten Schicht ist variabel in einem Bereich von 4 bis 112. Das SAND/1-PCI-Board ist für diesen Versuch mit vier SAND/1-Chips bestückt.

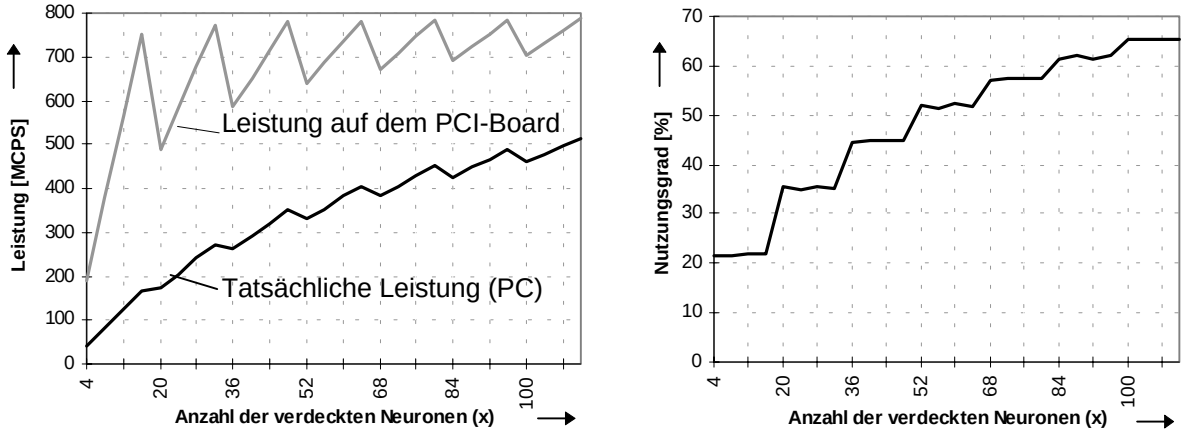


Abbildung 6.3: Leistung auf dem PCI-Board und dem PC für MLP1-Netze (512-x-16) (linkes Diagramm). Verhältnis zwischen beiden Leistungen (Nutzungsgrad, rechtes Diagramm).

Der prägnante Verlauf der auf dem Board erzielbaren Leistung tauchte bereits in Kapitel 4.3.5 bei der Betrachtung der Effizienz auf. Eine ähnliche Charakteristik ist auch bei der im PC gemessenen, tatsächlichen Leistung zu erkennen, wobei die Spitzenwerte stark gedämpft sind. Die Ursache ist darin zu suchen, daß vor allem bei der Berechnung kleiner Netze die Übertragungszeiten des PCI-Busses die Verarbeitungszeiten auf der SAND/1-PCI-Karte dominieren. Betrachtet man das Verhältnis zwischen beiden Leistungen (Abbildung 6.3, rechtes Diagramm), so sind deutlich Plateaus konstanter Breite und unterschiedlicher Höhen zu erkennen.

Die Breite der Plateaus beträgt genau 16 Neuronen und entspricht damit der Anzahl paralleler Prozeßelemente auf dem SAND/1-PCI-Board. Die Existenz dieser Plateaus läßt sich anschaulich erklären:

Zur Berechnung von $m[v]$ Neuronen in einer verdeckten Schicht v werden bei vier SAND/1-Chips

$$S[v] = \left\lceil \frac{m[v]}{16} \right\rceil \qquad (6.1)$$

Schritte benötigt. Wird die Anzahl der Neuronen erhöht, so nimmt alle 16 Neuronen die Anzahl der Schritte um einen Schritt zu. Dies hat zur Folge, daß einerseits die Verarbeitungszeit bis zum Erreichen dieser Schwelle nahezu konstant bleibt (Effizienz und Leistung nehmen zu). Andererseits müssen beim Überschreiten der Schwelle nochmals alle Aktivitäten der vorhergehenden Schicht erneut gelesen werden, so daß sich ein Sprung in der Verarbeitungszeit ergibt, die proportional zur Anzahl der Neuronen in der vorhergehenden Schicht (n) ist (6.2).

$$\Delta t_{calc} \cong 4 \cdot n \qquad (6.2)$$

Diese Charakteristik ist auch in der Kurve der theoretisch erzielbaren Leistung auf dem PCI-Board enthalten. Die gemessene Leistung beinhaltet noch einen zusätzlichen konstanten Zeitanteil der Datenübertragung. Mit jedem weiteren Schritt (alle 16 Neuronen) verliert dieser Anteil infolge gestiegener Rechenzeit auf dem Board an Gewicht, so daß sich eine quasi diskrete Abstufung des „Wirkungsgrades“ ergibt. Die Höhe dieser Stufen wird mit zunehmender Netzgröße geringer, insofern als, wie in diesem Beispiel, die Anzahl der über den PCI-Bus übertragenen Daten konstant bleibt.

Diese erste Untersuchung macht deutlich, daß der Wirkungsgrad sehr stark von der Netzgröße abhängt. In obigem Beispiel bewegte sich dieser in einem Bereich zwischen 22% und 65%. Je größer die berechneten Netze sind, um so günstiger ist das Verhältnis von Übertragungs- zur reinen Rechenzeit auf dem Board. Umgekehrt nimmt bei kleineren Netzen der Wirkungsgrad rapide ab, da die Übertragungszeit die gesamte Verarbeitungszeit dominiert.

Im folgenden sind zwei Beispiele dargestellt, die verdeutlichen, wie groß die Unterschiede im Nutzungsgrad ausfallen können.

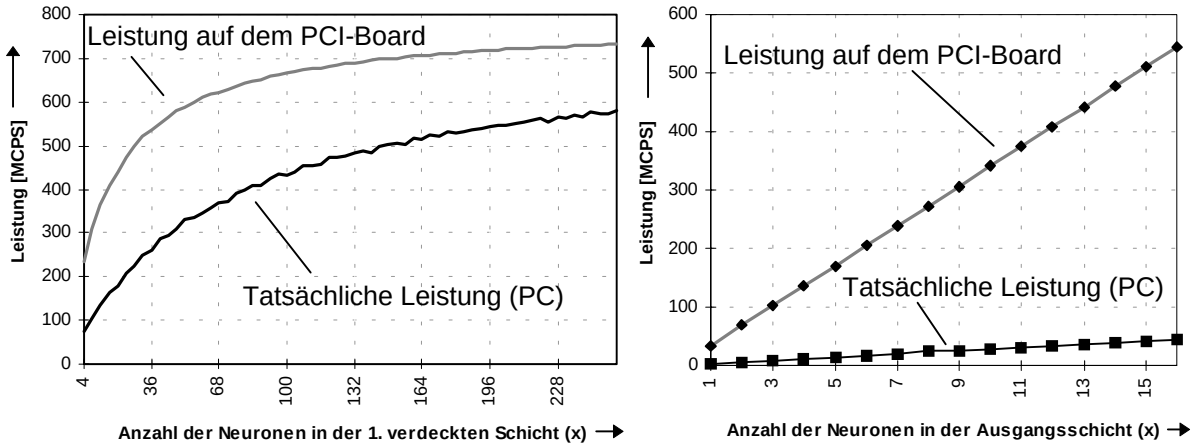


Abbildung 6.4: MLP2-Netz (4-x-240-4) mit hohem Nutzungsgrad (links) und MLP0-Netz (64-x) mit geringem Nutzungsgrad (rechts)

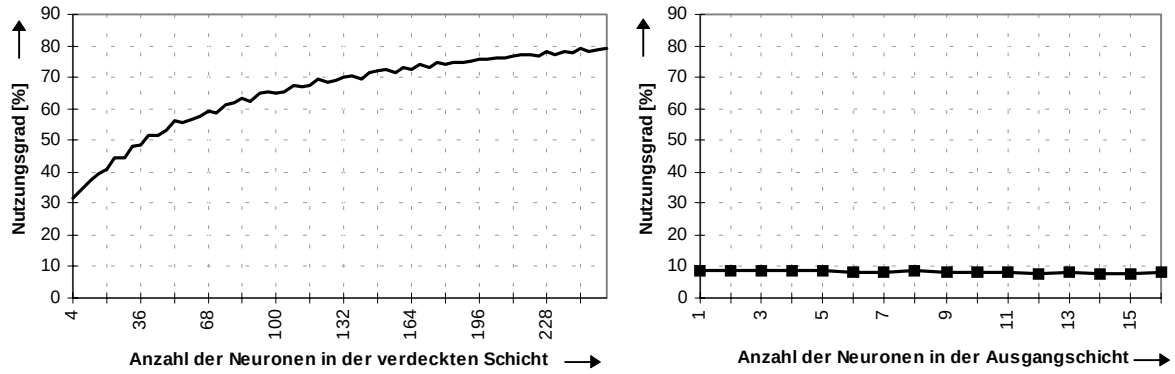


Abbildung 6.5: Verlauf der Nutzungsgrade für die beiden Netze aus Abbildung 6.4 (links MLP2, rechts MLP0)

In beiden Fällen war das SAND/1PCI-Board mit vier Neuro-Chips bestückt, die jedoch bei den betrachteten MLP0-Netzen (rechter Teil der Abbildung 6.4 und Abbildung 6.5) nur beim letzten (64-16) vollständig genutzt werden konnten. Der geringe Anteil der Rechenzeit an der Gesamt-Verarbeitungszeit führt zu einem Nutzungsgrad von durchschnittlich nur 8%. Völlig anders sieht die Situation bei den betrachteten MLP2-Netzen (linker Teil der Abbildung 6.4 und Abbildung 6.5) aus. Hier steigt der Nutzungsgrad auf 80% an und beträgt im Durchschnitt ca. 65%. Die höchste, in zahlreichen Versuchen ermittelte tatsächliche Leistung (PC) beträgt 614,4 MCPS.

Als Ergebnis dieser Untersuchungen kann ein grober Zusammenhang zwischen der Anzahl der Verbindungen in einem Netz und der tatsächlichen Leistung angegeben werden. Hierzu wurde die Leistung unterschiedlicher Netztopologien, die die gleiche Anzahl von Netzverbindungen besitzen, gemessen und gemittelt. Für Netzgrößen von 256 bis 65536 Verbindungen sind in Abbildung 6.6 die gemittelte Leistung und der Nutzungsgrad dargestellt.

Die Diskussion der hier aufgezeigten Probleme, eine quantitative Analyse und einige Vorschläge zur Behebung dieser Mängel erfolgt im Abschnitt 6.3.

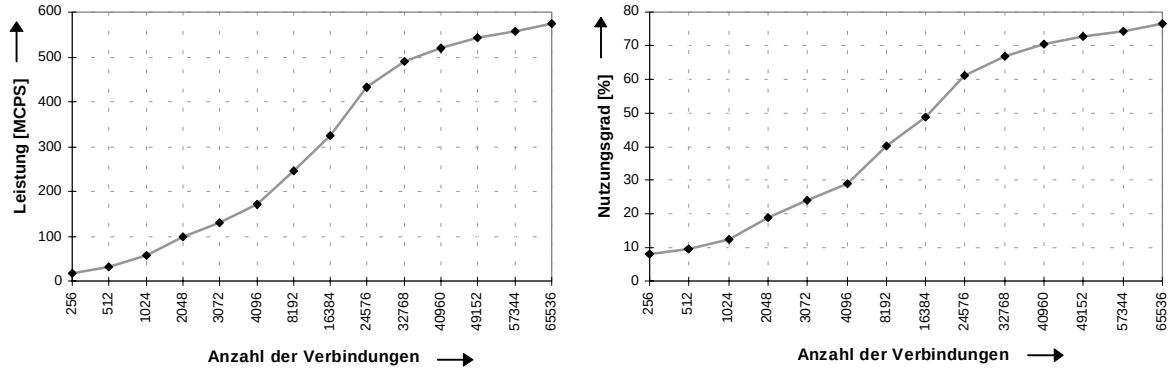


Abbildung 6.6: Leistung (links) und Nutzungsgrad (rechts) als Funktion der Gesamtzahl der Verbindungen eines Netzes bei der SAND/1-PCI-Karte

6.1.3 Beschleunigung gegenüber einer Software-Lösung

Ob der Einsatz von Hardware zur Beschleunigung neuronaler Operationen lohnt, hängt in erster Linie davon ab, wieviel schneller die parallele Hardware neuronale Algorithmen gegenüber einem sequentiellen Prozessor ausführen kann. Hierzu wurden, wie schon im Versuch zuvor, Netze unterschiedlicher Topologie betrachtet, die eine konstante Anzahl von Gewichten besitzen. Auf einem Pentium-II-Prozessor mit 266 MHz wurden für die entsprechenden Topologien die Leistungen gemessen und gemittelt. Für Netzgrößen von 256 bis 65536 Verbindungen ist im linken Diagramm der Abbildung 6.7 die durchschnittliche Leistung dargestellt. Der rechte Teil des Diagramms zeigt die Beschleunigung derselben Netze durch das SAND/1-PCI-Board.

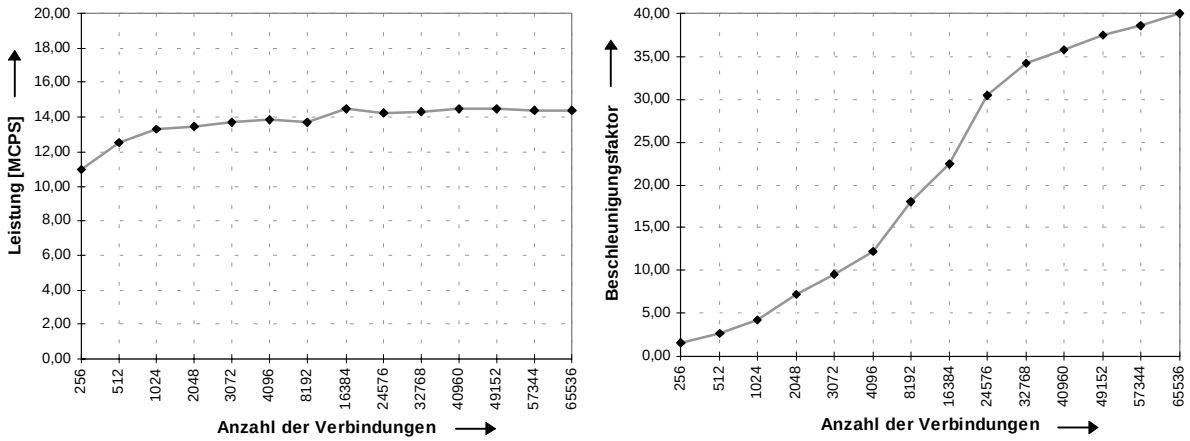


Abbildung 6.7: Leistung eines Pentium-II-Prozessors (Fließkommaarithmetik) mit 266 MHz (links) und die Beschleunigung durch das SAND/1-PCI-Board (rechts).

6.2 Vergleich mit leistungsfähigen Standardprozessoren

Im folgenden wird der SAND/1-Neuro-Chip mit zwei modernen und leistungsfähigen Prozessoren verglichen. Ziel dieser Untersuchungen ist es, genauere Erkenntnisse darüber zu gewinnen, inwiefern die wesentlich höhere Taktfrequenz eines Prozessors mit General-Purpose-Charakter Nachteile der Architektur (geringere Parallelisierung etc.) ausgleichen kann.

Stellvertretend für den PC-Sektor wird im folgenden die Beschleunigung neuronaler Operationen mit Hilfe der Architekturerweiterung MMX eines INTEL-Pentium-Prozessors untersucht. Für den Bereich der Stand-Alone-Anwendungen werden entsprechende Untersuchungen mit Hilfe des zur Zeit schnellsten DSPs, dem TMS320C6201 von Texas Instruments durchgeführt.

6.2.1 Vergleich mit INTEL-Pentium und Architekturerweiterung MMX

Die grundlegende Struktur der INTEL-Architekturerweiterung MMX wurde bereits in Kapitel 3.1.1 vorgestellt. Im Rahmen dieses Abschnitts soll anhand von Benchmarks die Leistungsfähigkeit untersucht und ein Vergleich zur SAND/1-Architektur gezogen werden.

Ergänzend zu den Betrachtungen in Kapitel 3.1.1 wird nachfolgend kurz das Prinzip der Beschleunigung von Algorithmen in der Art von Gleichung (4.1) durch die MMX-Erweiterung erläutert. Ausgangspunkt ist die vektorielle Darstellung der Operationen einer Schicht eines MLP-Netzes (4.3). Ähnlich wie bei der SAND-Architektur wird auch bei MMX die Gewichtsmatrix in Submatrizen unterteilt. Jede der beiden parallelen Pipelines (MMX-U und MMX-V, siehe Abbildung 3.1) multipliziert eine Submatrix der Größe (2 x 2) mit einem (2 x 1)-Vektor. Ein vereinfachtes Schema dieser Matrix / Vektor-Multiplikation mit den benötigten MMX-Instruktionen und -Registern ist in der Abbildung 6.8 dargestellt.

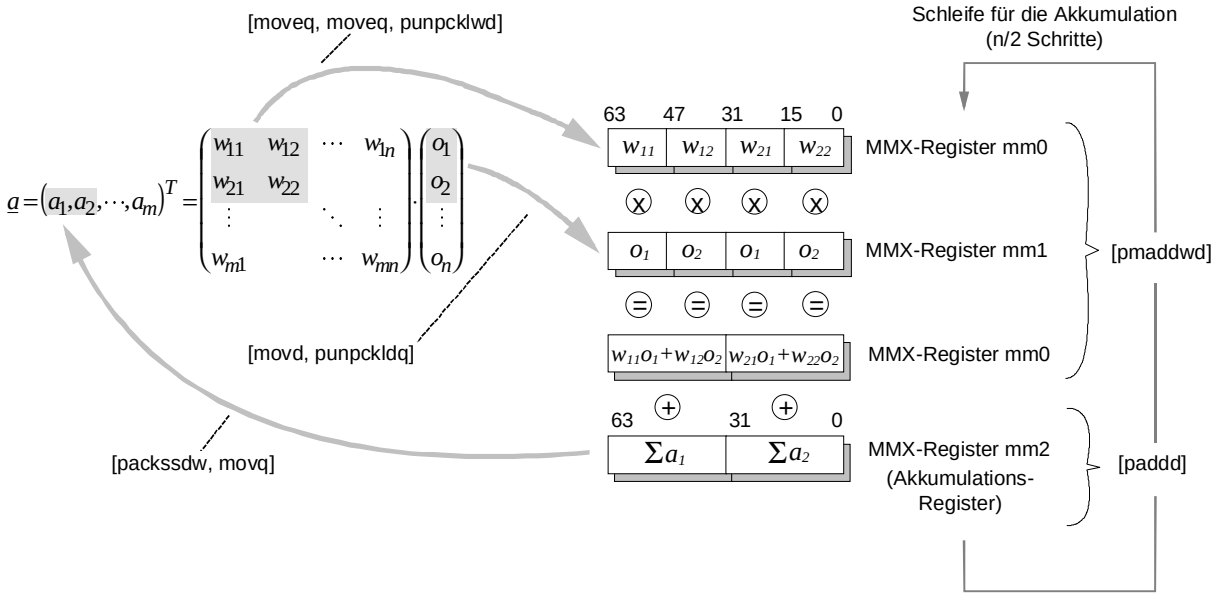


Abbildung 6.8: Vereinfachtes Schema der Matrix-Vektor-Multiplikation in einer MMX-Pipeline

Da sich die beiden MMX-Pipelines die Multiplizierereinheit teilen, ist es nicht möglich, in beiden Pipelines zeitgleich eine Matrix / Vektor-Multiplikation auszuführen. Eine geschickte Anordnung der Instruktionen ermöglicht es aber, daß Operanden einer Pipeline aus dem Speicher geladen werden, während die andere Pipeline die Multiplizierereinheit benutzt, so daß sich eine günstige Gesamtauslastung ergibt. Zu beachten ist, daß zunächst weder die Eingangsdaten (Aktivitäten) noch die Gewichte im Cache stehen. Jeder Cache-Ladevorgang dauert minimal 11 Takte, so daß der Datentransfer insgesamt die reine Berechnungszeit dominieren würde. Berücksichtigt man, daß beim Zugriff auf eine Adresse im Speicher der Cache-Controller eine gesamte Cacheline (32 Bytes) [Intel98a] in den Cache

lädt, so kann die Anzahl der langsamen Speicherzugriffe deutlich reduziert werden, wenn die bereits im Cache stehenden Daten genutzt und nicht zu einem späteren Zeitpunkt erneut geladen werden. Bezogen auf das vereinfachte Schema in Abbildung 6.8 bedeutet dies, daß beim Lesen des Gewichtes $w_{1,1}$ aus dem Speicher die nachfolgenden 15 Gewichte ($w_{2,1}$ ... $w_{16,1}$) ebenfalls im Cache gespeichert werden, sofern diese im 16-Bit-Integer-Format vorliegen. Um diese Daten zu nutzen, sind vier Akkumulatoren erforderlich, von denen jeder wiederum vier Zeilen der Matrix $\underline{W}$ verarbeitet, so daß insgesamt in einem überlappenden Schema 16 Summen gebildet werden, die jeweils paarweise durch eine der beiden MMX-Pipelines (U, V) parallel berechnet werden. Während eine Pipeline eine Multiplikation ausführt (pmaddwd) werden über die zweite Pipeline die nächsten Operanden geladen und in das entsprechende MMX-Format (punpackldq) konvertiert, Teilsummen akkumuliert (padd) bzw. Zwischenergebnisse aus einem der MMX-Register zurück in den Speicher geschrieben (movq).

Für die Operationen einer Schicht in einem MLP-Netz hat diese Optimierung zur Folge, daß die Effizienz besonders dann hoch ist, wenn die Anzahl der Neuronen einer (Sende-) Schicht ν ein Vielfaches von zwei und die Anzahl der Neuronen in der (Empfangs-) Schicht ν+1 ein Vielfaches von 16 ist, da durch das sogenannte Loop-Unrolling, zur Nutzung einer gesamten Cacheline (32 Bytes), stets 16 Gewichte in den Cache geladen werden.

Trotz dieser Optimierung ist es nicht möglich, die volle Leistung der MMX-Architektur zu nutzen, da infolge des partiellen Loop-Unrollings (16 Akkumulatoren) nicht genügend MMX-Register zur Verfügung stehen. Dies hat zur Folge, daß Zwischenergebnisse in den Speicher ausgelagert werden müssen. Da auf die entsprechenden Adressen ständig zugegriffen wird, kann davon ausgegangen werden, daß sich die Daten im schnellen 1st-Level-Cache (L1) im Prozessor befinden. Dennoch dauert ein L1-Cache-Zugriff im günstigsten Fall drei Takte, während für einen Registerzugriff nur ein Takt zu veranschlagen ist.

Unter Berücksichtigung der Latenzzeiten für den MAC-Befehl (pmaddwd) und das Laden von Daten aus dem Speicher (was infolge des Loop-Unrollings unumgänglich ist), ergibt sich eine verzahnte Verarbeitung der 16 Summen in einer Schleife.

Ein wesentlicher Unterschied zwischen der Beschleunigung neuronaler Netze durch die SAND/1-Architektur und der MMX-Erweiterung besteht in der Größe der Operanden. Bei SAND/1 werden sowohl die Gewichte als auch die Aktivitäten mit einer Wortbreite von 16 Bit verarbeitet. Die Arithmetikeinheiten und die Register wurden hierfür dimensioniert (Kapitel 4.4.3). Insbesondere ergab sich für die Akkumulation von bis zu 512 Werten eine Akkumulator-Wortbreite von 40 Bit. Bei MMX beträgt die maximale Wortbreite eines Akkumulator-Registers jedoch nur 32 Bit. Geht man wie bei der Dimensionierung der Arithmetikeinheiten und Register bei SAND/1 von Schichten mit maximal 512 Neuronen aus, so ergibt sich hieraus, daß einer der Operanden nur im 8-Bit-Format verarbeitet werden kann. Die nachfolgenden Leistungsangaben wurden deshalb entsprechend der Gleichung (2.21) angepaßt.

Zur Ermittlung der Verarbeitungsleistung wurde eine Bibliothek für Windows95 / NT 4.0 zur Berechnung von MLP-Netzen auf einem Pentium-Prozessor mit MMX-Erweiterung entwik-

kelt. Auf der Basis dieser Bibliothek wurde mit Hilfe eines Counters[18] die Verarbeitungszeit der verschiedenen Netze gemessen.

Anhand eines MLP1-Netzes (512-x-6), das bereits in Kapitel 6.1.2 zur Bewertung der SAND/1-Architektur herangezogen wurde, lassen sich einige Eigenschaften der MMX-Architektur erläutern. Die Leistung, aufgetragen über der Anzahl der Neuronen in der verdeckten Schicht, zeigt einen stark zerklüfteten Verlauf (Abbildung 6.9).

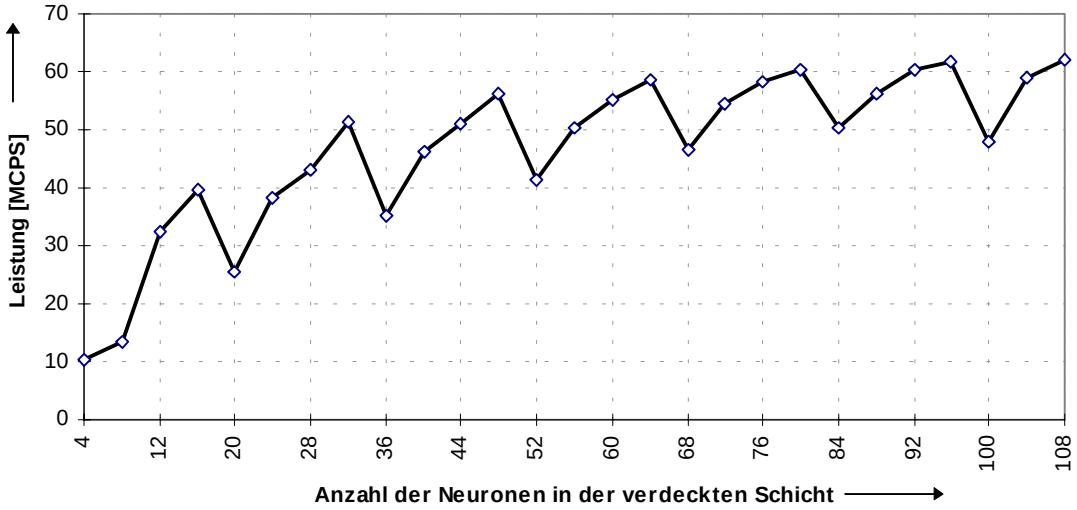


Abbildung 6.9: MLP1-Netz (512-x-16) berechnet auf einem Pentium-II-Prozessor (266 MHz) unter Verwendung der MMX-Architekturerweiterung. Verlauf der Leistung [MCPS] über der Anzahl der Neuronen in der verdeckten Schicht.

Der Abstand der Spitzenwerte in horizontaler Richtung ist konstant und beträgt 16 Einheiten auf der Abszisse. Der charakteristische Verlauf der Leistung ist eine Folge der Optimierung der Cache-Ausnutzung, da die Gewichtsmatrix in Submatrizen der Größe (16 x 2) unterteilt wird. Ähnlich wie bei der SAND/1-Architektur steigt die Leistung mit jedem weiteren Neuron in der zu berechnenden Schicht an. Übersteigt die Anzahl der Neuronen den Wert 16 oder ein ganzzahliges Vielfaches davon, dann wird die Gewichtsmatrix um 16 Zeilen erweitert, von denen im ungünstigsten Fall nur eine benötigt wird. Als Folge ergibt sich eine Leistungsminderung unmittelbar an solchen Übergängen.

Eine allgemeinere Aussage ist möglich, wenn die Leistung über der Anzahl der Verbindungen aufgetragen wird. Hierzu wird aus verschiedenen Netztopologien mit gleicher Anzahl von Verbindungen die Leistung gemessen und gemittelt. Für MLP-Netze mit 256 bis 65536 Verbindungen ist der Verlauf der Leistung eines Pentium-II-Prozessors (266 MHz) unter Verwendung der MMX-Erweiterung im linken Diagramm der Abbildung 6.10 dargestellt. Das rechte Diagramm zeigt den Beschleunigungsfaktor des SAND/1-PCI-Boards gegenüber dem Pentium-II-Prozessor mit MMX-Erweiterung.

[18] Es handelt sich um einen sogenannten „Time Stamp Counter“ des Pentium-Prozessors, der in jedem Takt inkrementiert wird und mittels RDTSC gelesen werden kann [Intel98a].

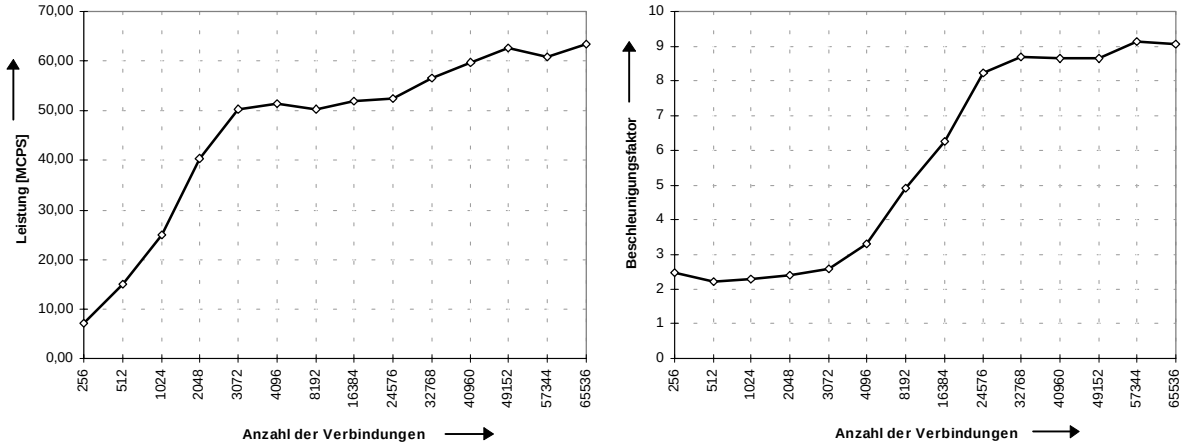


Abbildung 6.10: Verlauf der Leistung bei der Berechnung von MLP-Netzen auf einem Pentium-II-Prozessor (266 MHz) unter Verwendung der Architekturerweiterung MMX (links). Im Vergleich hierzu: Beschleunigung durch das SAND/1-PCI-Board (rechts)

Bei der Durchführung der Benchmarks wurde der Einfluß der Cache-Speicher (Daten) auf die Gesamtleistung deutlich. Bei der ersten Netzberechnung werden sämtliche Daten (Gewichte und Aktivitäten) aus dem Hauptspeicher geladen und dabei auch im Cache abgelegt. Somit kann die zweite Netzberechnung ca. 10% schneller durchgeführt werden, da zumindest die Gewichtsmatrix nicht mehr aus dem (langsamen) Hauptspeicher geladen werden muß.

Sowohl in Abbildung 6.9 als auch in Abbildung 6.10 ist die geringe Leistung bei kleinen Netzen sehr auffällig. Diese Leistungseinbuße kann aufgrund der Architektur von MMX nicht erklärt werden, d.h. auch bei kleinen Netzen müßte eine wesentlich höhere Leistung möglich sein. Die Messung der Verarbeitungszeit der durch MMX parallelisierten Matrix-Vektor-Multiplikation (Assembler) brachte zu Tage, daß diese Zeit einen konstanten Anteil enthält, der offensichtlich von der Größe der Operanden unabhängig ist. Hierzu wurde ein MLP0-Netz mit zwei bis 128 Neuronen in der Eingangsschicht und 16 Neuronen in der Ausgangsschicht untersucht. Unter Verwendung eines Pentium-Prozessors mit 166 MHz ergab sich für die Netze von zwei bis 40 Eingangsneuronen eine konstante Verarbeitungszeit von ca. 20µs. Erst bei mehr als 40 Eingangsneuronen stieg die Verarbeitungszeit bis auf 58 µs (128 Eingangsneuronen) an. Mit den zur Verfügung stehenden Software-Werkzeugen war es nicht möglich, genauere Informationen über diesen Sachverhalt zu gewinnen. Eine mögliche Ursache könnte die Sicherung der sowohl von MMX als auch von der Floating-Point-Einheit genutzten Register vor, bzw. nach dem Einsatz der MMX-Einheit sein.

Neben dieser Leistungsminderung bei kleinen Netzen, sind der MMX-Technologie vor allem durch den Zugriff auf den, im Vergleich zum Prozessor, langsamen Speicher Grenzen gesetzt. Um den Cache optimal zu nutzen, ist ein partielles Loop-Unrolling erforderlich, wofür jedoch nicht genügend MMX-Register zur Verfügung stehen. Ein weiterer Nachteil ist die geringe Akkumulator-Wortbreite von lediglich 32 Bit. Eine ausführliche Diskussion dieser Ergebnisse erfolgt im Kapitel 6.3.

6.2.2 Vergleich mit dem digitalen Signalprozessor TMS320C6201

Für die schnelle digitale Signalverarbeitung haben sich digitale Signalprozessoren in vielen Bereichen etabliert. Einer der zur Zeit schnellsten DSPs mit Festkommaarithmetik ist der TMS320C6201 von Texas-Instruments, dessen Architektur in groben Zügen bereits in Kapitel 3.1.2 vorgestellt wurde. Durch seine hohe Integrationsdichte eignet er sich vor allem für Stand-Alone-Anwendungen, kann aber auch in Verbindung mit einem PCI-Board zur Beschleunigung von PC-Applikationen eingesetzt werden. Einer der ersten Hersteller einer solchen Einsteckkarte ist die Firma Loughborough Sound Images, deren PCI/C6200-Karte für die folgenden Benchmarks eingesetzt wurde. Ergänzend zu der Einführung in Kapitel 3.1.2 wird im folgenden zunächst kurz erläutert, wie mit Hilfe der C60-Architektur MLP-Netze beschleunigt werden können.

Ausgangspunkt der folgenden Betrachtungen ist wiederum die vektorielle Darstellung der Gleichung (4.3). Während sich bei SAND/1 und MMX eine Beschleunigung durch die parallele Berechnung mehrerer Neuronen der jeweiligen Schicht ergibt, wird beim C6201 die Akkumulation parallelisiert. Das bedeutet, daß mit Hilfe des C6201 die Vektor / Vektor-Multiplikation als Bestandteil der Matrix / Matrix-Multiplikation beschleunigt wird. Da maximal zwei Multiplikationen parallel in einem Takt ausgeführt werden können [Texas97a], werden theoretisch bei n Neuronen in der vorhergehenden Schicht $n/2$ Takte für die Berechnung eines Neurons in der zu berechnenden Schicht benötigt. Der Grund für diese Form der Parallelisierung liegt sowohl in der Speicherarchitektur als auch in der Pipeline-Struktur des C6201, die anhand der folgenden Abbildung 6.11 erläutert werden sollen.

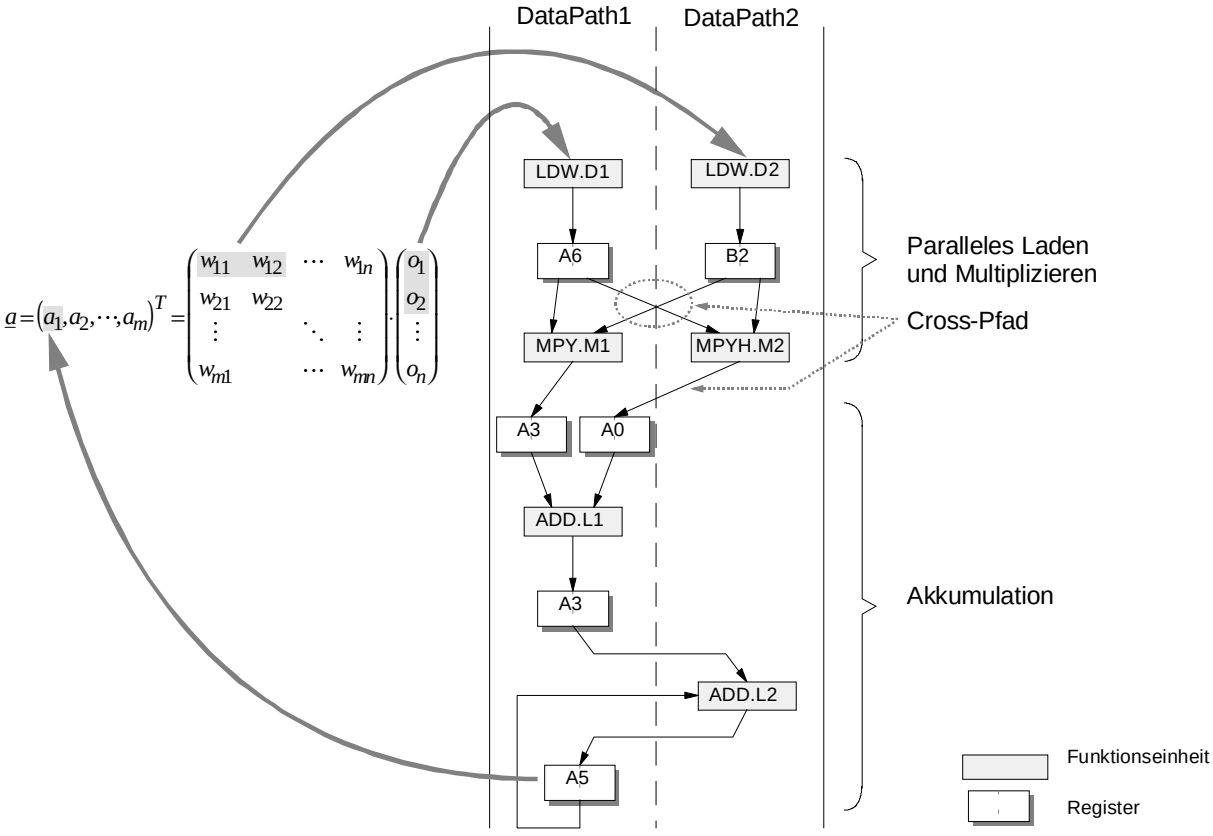


Abbildung 6.11: Vereinfachtes Schema der Vektor-Vektor-Multiplikation beim TMS320C6201

Die Organisation des internen Datenspeichers erlaubt den gleichzeitigen Zugriff auf zwei 32-Bit-Wörter, sofern diese nicht in derselben Speicherbank liegen. Mittels LDW (**L**oa**dW**ord) können zwei im Speicher benachbarte 16-Bit-Wörter in ein 32-Bit-Register geladen werden. Durch die Möglichkeit, High- und Low-Word parallel zu verarbeiten, ergeben sich so zwei Multiplikationen pro Takt (Low-Word: MPY.M1, High-Word: MPYH.M2). Durch die Verwendung von Registern kann eine Pipelinestruktur aufgebaut werden, so daß im gleichen Takt noch ein Akkumulationsschritt (verteilt auf zwei Pipelinestufen, siehe Abbildung 6.11) durchgeführt werden kann (ADD.L1, ADD.L2). Zu beachten ist, daß pro Takt nur ein Cross-Pfad in jede Richtung zu Verfügung steht, d.h. nur eine der funktionalen Komponenten darf einen ihrer Operanden vom Registerfile des anderen Datenpfades lesen. In obigem Beispiel wurde der Cross-Pfad bei den beiden Multiplizieren (.M1 und .M2) verwendet, da nur so das Laden und Multiplizieren parallelisiert werden kann. Der Grund liegt darin, daß mit einem Load-Befehl (LDW) zwei 16-Bit-Operanden gleichzeitig in ein 32-Bit-Register (A6 und B2) geladen werden. Die Parallelisierung erfordert nun, daß jeweils einer dieser Operanden im anderen Datenpfad verarbeitet werden muß. Das bedeutet, daß der Multiplizierer im ersten Datenpfad (MPY.M1) die Low-Words der beiden Register A6 und B2 multipliziert, während gleichzeitig der Multiplizierer im zweiten Datenpfad (MPYH.M2) die beiden High-Words von A6 und B2 multipliziert.

Um zwei Vektoren vollständig miteinander zu multiplizieren, muß das in Abbildung 6.11 dargestellte Schema mehrfach in einer Schleife wiederholt werden. Ohne hier allzusehr auf die Details einzugehen, sei dennoch erwähnt, daß dies ein partielles Loop-Unrolling zur Folge hat. Ähnlich wie bei MMX müssen beim C6201 zwei der oben dargestellten Schemata in einer Schleife angeordnet werden. Vor der eigentlichen Schleife, bis zum Füllen der Pipeline, ergibt sich ein sogenannter Loop-Prolog (neun Takte), im Anschluß an die Schleife folgt ein sogenannter Loop-Epilog (zehn Takte). Beide führen dazu, daß, bezogen auf die gesamte Schleife, nicht die theoretische Leistung von zwei MAC-Operationen pro Takt erreicht werden kann. Jedoch nimmt der Einfluß dieser beiden Teile mit zunehmender Vektorgröße ab.

Der Einsatz und die Zuordnung der insgesamt acht parallelen Funktionseinheiten (Abbildung 3.2) erfolgt im Rahmen einer sogenannten VLIW (**V**ery **L**ong **I**nstruction **W**ord)-Steuerung. Mit einem einzigen Instruktionswort wird das Verhalten aller acht Module während der nächsten Takte gesteuert. Für jedes Modul stehen hierfür 32 Bit zur Verfügung. Dieser Mechanismus ist im allgemeinen dem Anwender nicht zugänglich, da der Designflow die Entwicklung der Programme in C vorsieht. Mit einem optimierenden Compiler wird dann ein Algorithmus, soweit möglich, parallelisiert, so daß schließlich der Compiler die VLIW-Befehlsworte generiert. Danach ist es kaum mehr möglich, im Maschinen-Code noch Teile des ursprünglichen Algorithmus zu identifizieren. Aus diesem Grunde wurde die Multiply-Accumulate-Schleife in einem sogenannten Linearen Assembler (SA) geschrieben. Hierbei werden unmittelbar Assembler-Befehle verwendet, deren exaktes Scheduling aber nach wie vor vom Compiler vorgenommen wird.

In einem ersten Versuch wurde untersucht, wie viele 16/16-Bit-MAC-Operationen der C6201 tatsächlich pro Sekunde ausführen kann. Hierzu wurden MLP0-Netze mit 16 bis 512 Neuronen in der Eingangsschicht berechnet, und es wurde die Leistung ermittelt. Bei einer Taktfrequenz von 160 MHz und zwei MAC-Operationen pro Takt ergibt sich ein theoreti-

scher Maximalwert von $320 \cdot 10^6$ MAC-Operationen pro Sekunde (MMAC/s). Die tatsächliche (gemessene) Leistung für die betrachteten Netzgrößen ist in der Abbildung 6.12 dargestellt.

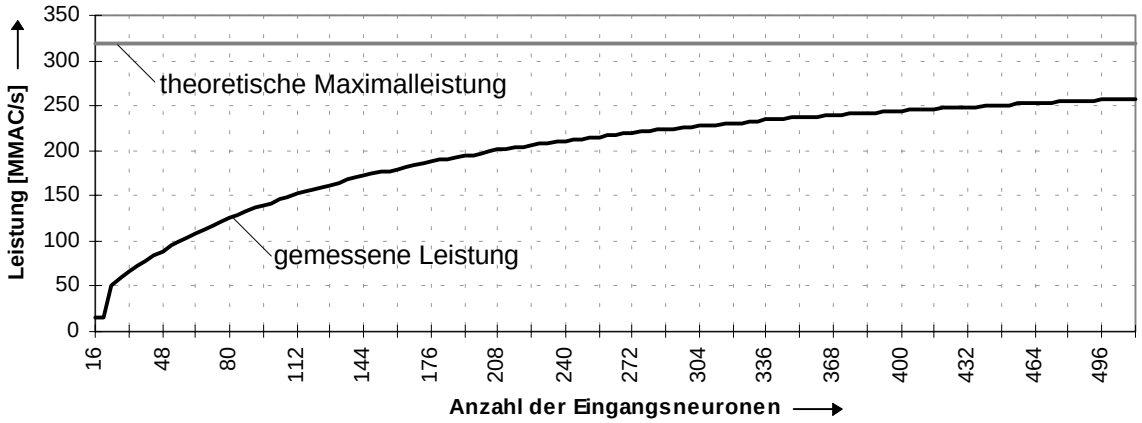


Abbildung 6.12: Anzahl der MAC-Operationen pro Sekunde beim C6201, aufgetragen über der Anzahl der Eingangsneuronen

Die Abweichung zum theoretischen Maximalwert rührt vom Schleifen-Prolog und -Epilog her, deren Einfluß mit zunehmender Netzgröße zwar abnimmt, aber auch noch bei 512 Eingangsneuronen deutlich zu erkennen ist.

Die in Abbildung 6.12 dargestellte Leistungskurve stellt die Obergrenze der erzielbaren Gesamtleistung dar, da noch keine Transferzeiten zu externen Speichern berücksichtigt wurden. Sowohl Gewichte als auch Aktivitäten wurden beim obigen Versuch aus dem internen Datenspeicher des C6x gelesen. Insbesondere große Netze erfordern, daß die Gewichte außerhalb des DSPs in einem separaten Gewichtsspeicher verwaltet werden, da der interne Speicher mit 64 KBytes hierfür zu klein ist. Im kontinuierlichen Betrieb müssen Ein- und Ausgangsaktivitäten von bzw. zu externen Speichern transportiert werden. Ein dritter Aspekt, der beim obigen Versuch unberücksichtigt blieb, ist die Berechnung der Transferfunktion, die sowohl bei SAND/1 als auch bei MMX in Form einer Look-Up-Tabelle realisiert wurde.

Unter Verwendung der bereits erwähnten PCI-Karte PCI/C6200 und der hierauf verfügbaren Speicher kann die Leistungsfähigkeit des C6201 in einer realen Umgebung untersucht und bewertet werden. Dabei wird die Karte zunächst als Stand-Alone-System betrachtet, so daß die Übertragung von Daten über den PCI-Bus bei den folgenden Untersuchungen unberücksichtigt bleibt. Ein vereinfachtes Schema des C6201 und der ihn umgebenden Speicher ist in Abbildung 6.13 dargestellt.

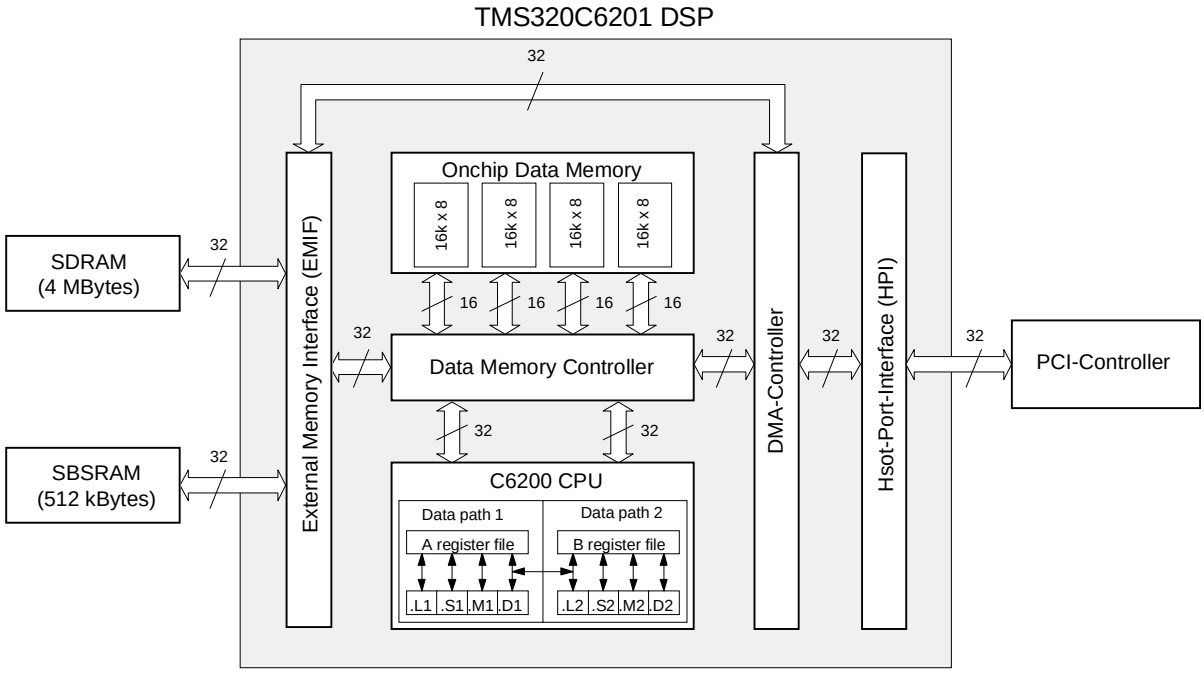


Abbildung 6.13: Speicher, PCI-Controller und C6201 DSP auf der PCI/C6200 Karte

Die maximale Leistung von zwei MAC-Operationen pro Takt kann nur dann erzielt werden, wenn simultan vier Operanden aus dem Speicher gelesen werden können. Wird für die Gewichte und die Aktivitäten der interne Speicher des C6201 verwendet (64 KBytes), dann kann diese Forderung erfüllt werden, da sowohl der Memory-Controller als auch der interne Speicher zwei simultane 32-Bit-Zugriffe der C6200-CPU erlauben. Völlig anders sieht die Situation bei Zugriffen auf die externen Speicher aus. Hier kann immer nur ein Operand pro Takt gelesen und entweder im internen Speicher oder in einem Register der CPU abgelegt werden. Zusätzliche Verzögerungen ergeben sich durch die externen Speicher, die je nach Art der Übertragung Wartezyklen verursachen. Aus diesen Gründen scheinen direkte CPU-Zugriffe auf externe Speicher wenig sinnvoll zu sein.

Um dennoch Daten von außerhalb verarbeiten zu können, müßten diese, wenn auch mit geringerer Übertragungsrate, während einer laufenden Operation übertragen und im internen Speicher des DSP abgelegt werden. Hierfür kann der DMA-Controller verwendet werden, der unabhängig von der CPU Daten im gesamten Speicherbereich des DSPs bewegen kann. Voraussetzung ist, daß weder bei Lese- noch bei Schreibzugriffen Konflikte auftreten. Da die CPU jedoch in jedem Takt vier Operanden aus dem internen Speicher liest, werden hierbei alle vier Speicherbänke durch Leseoperationen blockiert, so daß während einer laufenden MAC-Schleife der DMA-Controller keinen Schreibzugriff auf den internen Speicher des C6201 durchführen kann. Somit muß die Übertragung von Gewichten und Aktivitäten von externen Speichern in den internen Speicher des C6201 durch die CPU selbst durchgeführt werden. Während Daten vom externen in den internen Speicher übertragen werden, kann die CPU keine (Rechen-) Operationen ausführen. Hinzukommt, daß die (gemessene) Übertragungsrate zwischen externem und internem Speicher nur ca. 39 MBytes/s beträgt. Für ein Netz mit 512 Eingangsneuronen ergibt sich eine Übertragungszeit der Aktivitäten von 26,25 µs bei einer Verarbeitungszeit (512 MAC-Operationen) von 1,56µs. Damit ist höchstens noch eine Gesamtleistung von 18 MCPS möglich.

Die resultierende Gesamtleistung für MLP0-Netze mit 16 Neuronen in der Ausgangsschicht und 16 bis 512 Neuronen in der Eingangsschicht ist in Abbildung 6.14 dargestellt. Bei diesem Versuch wurde die Transferfunktion als Look-Up-Tabelle im externen SBSRAM abgelegt. Die Gewichtsmatrix, welche ebenfalls im SBSRAM gespeichert ist, wird mit Hilfe der MEMCOPY-Funktion, vor Beginn der eigentlichen Berechnung, zeilenweise in den internen Speicher des C6201 kopiert.

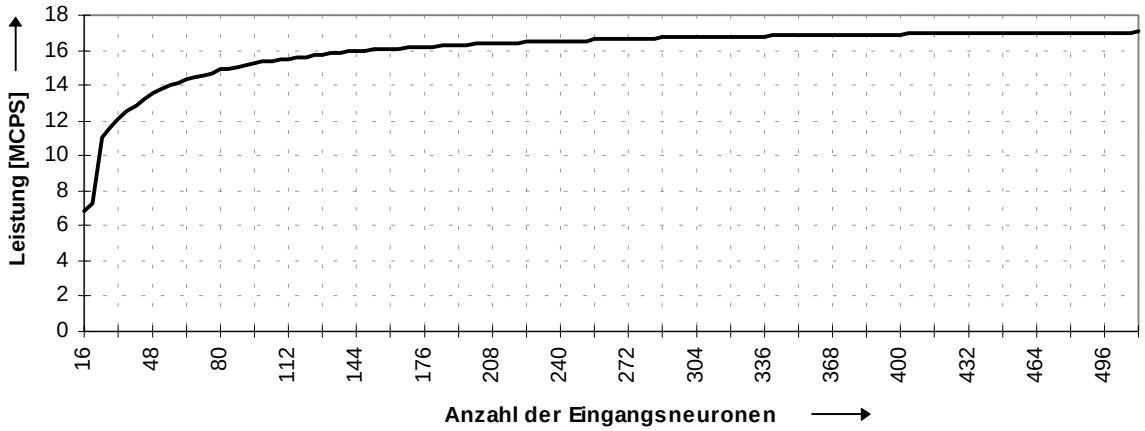


Abbildung 6.14: Gesamtleistung des TMS320C6201 in MCPS bei der Berechnung von MLP0-Netzen (x-16), aufgetragen über der Anzahl der Eingangsneuronen (x)

Angesichts der geringen Gesamtleistung des C6201 und der Tatsache, daß das verwendete PCI-Board PCI/C6200 keine PCI-Burst-Transfers unterstützt, macht es keinen Sinn, die im PC zur Verfügung stehende Leistung zu ermitteln und mit der von MMX bzw. SAND/1 zu vergleichen, da diese höchstens in der Größenordnung der Fließkommaleistung eines Pentium-II-Prozessors liegen wird.

Die Untersuchungen verdeutlichen, daß der C6201 nur dann eine hohe Leistung entfalten kann, wenn sämtliche Operanden im internen Speicher stehen. Dann ist es, zumindest bei langen Akkumulationszyklen möglich, bis zu 270 MMAC/s zu erreichen. Weitere Versionen dieses DSPs sollen laut Herstellerangaben mit bis zu 250 MHz getaktet werden können, so daß hierdurch eine maximale Leistung von 400 MMAC/s möglich wird. Als Flaschenhals entpuppt sich der Zugriff auf externe Speicher, der bei realen Anwendungen und großen Netzen unumgänglich ist. Die optimale Auslastung der CPU hat zur Folge, daß Lesezugriffe auf den internen Speicher keinen DMA-Transfer im Hintergrund zulassen. Abhilfe könnte hier eine neue Version des DSPs, der TMS320C6201B, schaffen, der zwei Speicherblöcke zu je vier Bänken besitzt (Abbildung 6.15). Dadurch können sich CPU-Operationen in der Art von Abbildung 6.11 und DMA-Transfers überlappen. Nach wie vor bleibt aber der 32-Bit-Bus zu externen Speichern als weiterer Flaschenhals bestehen. Erst, wenn bei größerem internen Speicher die Gewichtsmatrix im DSP abgelegt werden kann, wird es möglich sein, die volle Leistung der C6200-CPU auszuschöpfen. Geht man von einer Taktfrequenz von 250 MHz und einem genügend großen internen Speicher aus (mindestens 128 KBytes), so könnte eine maximale Leistung von 400 MCPS realistisch sein.

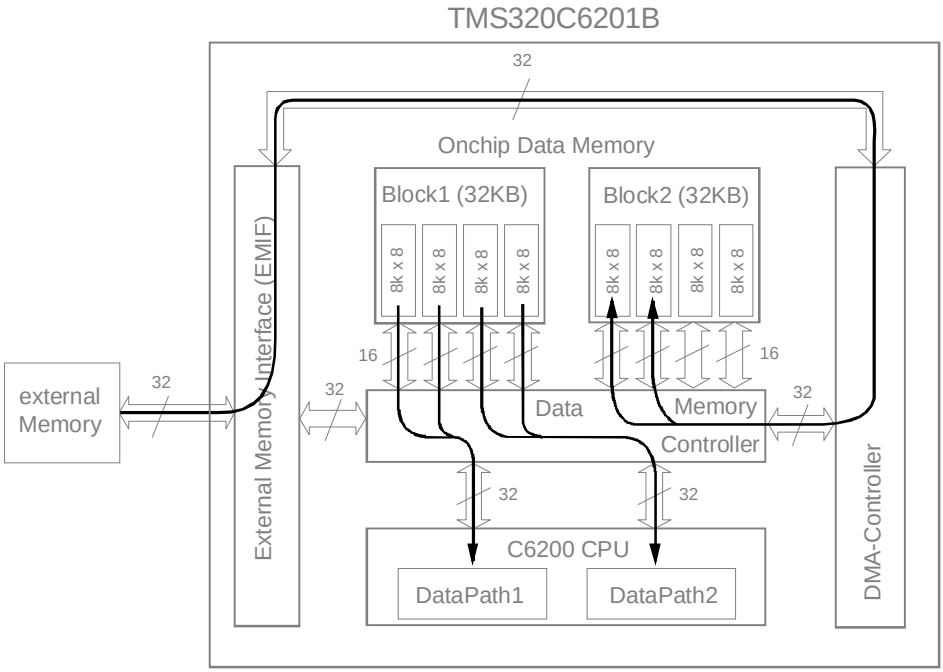


Abbildung 6.15: Digitaler Signalprozessor TMS320C6201B mit zwei unabhängigen internen Speicherblöcken. Die eingezeichneten Pfeile stellen die parallelen Speicherzugriffe (DMA und CPU) dar

6.3 Diskussion und Auswertung der Ergebnisse

Datenfluß auf dem PCI-Board

Bei der Konzeption von SAND/1 wurde berücksichtigt, daß sich die beiden Datenströme (Ein- und Ausgangsdaten) überlagern können, so daß der Transport der Daten die Leistung nicht beeinflußt. Voraussetzung ist, daß die Sendeschicht mindestens vier Neuronen besitzt. Für Schichten, die dieses Kriterium erfüllen, ist sichergestellt, daß das Leeren der internen Registerbänke abgeschlossen ist, bevor die folgenden Ergebnisse aus den ALUs dort abgelegt werden. Vor allem dann, wenn eine Schicht mehr Neuronen besitzt als parallele Prozeßelemente vorhanden sind, ergibt sich ein kontinuierlicher Datenstrom, so daß sich bei vier parallelen Prozeßelementen und 50 MHz Taktfrequenz eine tatsächliche Gesamtleistung von 200 MCPS pro SAND/1 einstellt. Bei der Betrachtung eines gesamten Netzes ergeben sich nur dann leistungsmindernde Leertakte, wenn (bei einem SAND/1) eine verdeckte Schicht nur vier Neuronen besitzt. In diesem Fall kann die Pipeline (Transportweg vom Ausgang des SAND/1 über die Look-Up-Tabelle und den FIFO_B zum Eingang des SAND/1) während der laufenden Berechnung nicht gefüllt werden, so daß die hieraus resultierende Latenzzeit einmalig mit sechs Takten zu Buche schlägt, was angesichts mehrerer Akkumulationszyklen von bis zu 2048 Takten einen verschwindend kleinen Verlust darstellt.

Das Zeitdiagramm in Abbildung 6.17 zeigt anhand eines konkreten Beispiels (4 x 12 x 4 MLP0-Netz) den resultierenden Datenfluß. Zu beachten ist, daß in jedem Takt eine Aktivität verarbeitet wird (act_in). Für die Berechnung der verdeckten Schicht (Neuronen N5-N16) sind mit einem SAND/1-Chip S=3 Schritte erforderlich. Zum besseren Verständnis der zeitlichen Zusammenhänge ist in Abbildung 6.16 ein vereinfachtes Blockdiagramm, bestehend aus dem Neuro-Chip-SAND/1, der Look-Up-Tabelle und einem FIFO, dargestellt.

(a) Vereinfachtes Schema des Datenflusses (b) Beispiel-Netz (4 x 12 x 4)

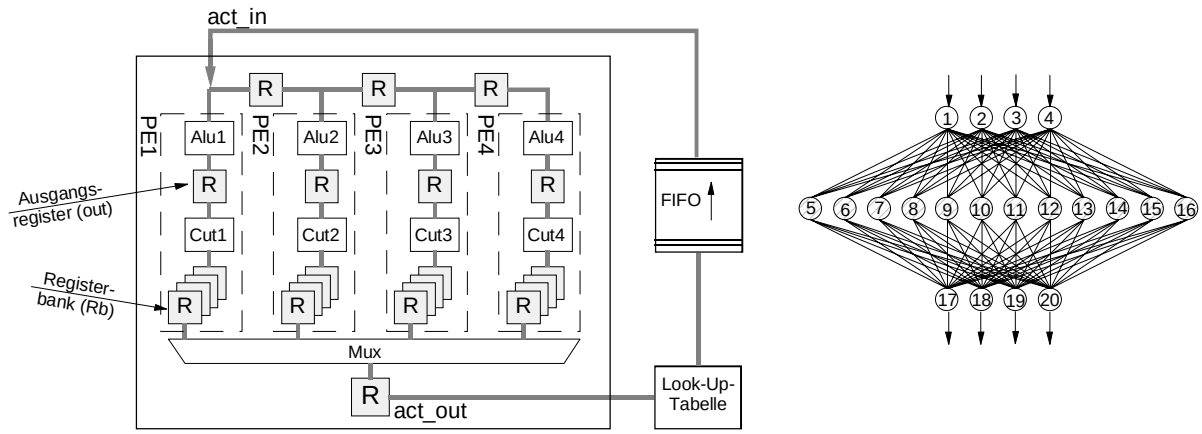


Abbildung 6.16: Vereinfachtes Schema des Datenflusses (a) und Beispiel-Netz (b) zur Demonstration des optimierten Datenflusses

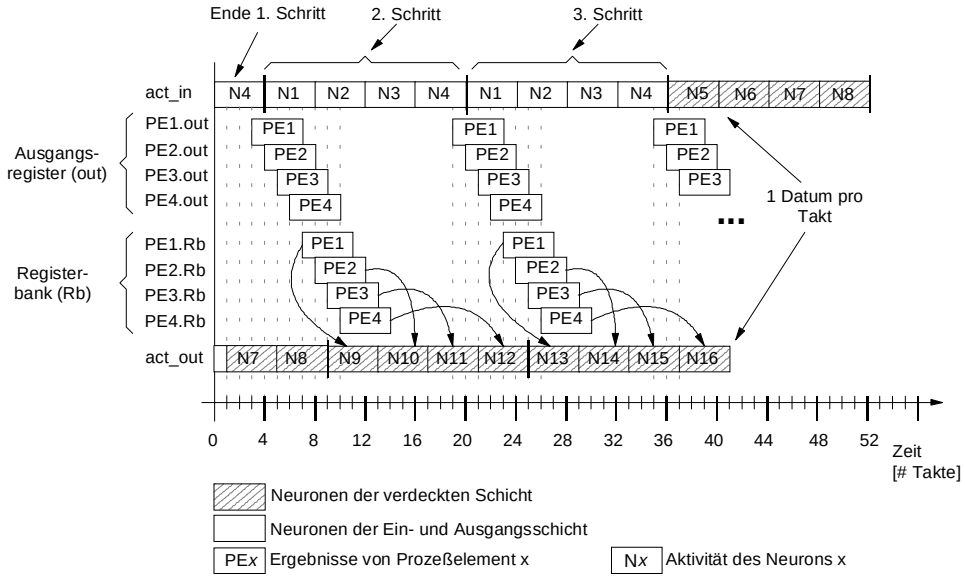


Abbildung 6.17: Zeitdiagramm für das Netz (4 x 12 x 4) und die Hardware-Konfiguration (1 SAND/1) der obigen Abbildung

Sowohl der optimierte Datenfluß innerhalb des SAND/1-Chips als auch dessen optimierte Anbindung an externe Speicher (Rückkopplungsschleife SAND/1 - Look-Up-Tabelle - FIFO - SAND/1) ermöglichen eine tatsächliche Leistung, die bis auf wenige Ausnahmefälle (s. oben) der theoretischen Leistung entspricht. Damit wurde, zumindest was den Datenfluß betrifft, ein Höchstmaß an Optimierung erreicht.

Einflüsse des Steuerwerks

Messungen auf dem SAND/1-PCI-Board zeigen (Abbildung 6.1), daß zwischen der oben skizzierten und der auf dem Board verfügbaren Leistung eine Lücke klafft, wenngleich die resultierende Gesamtleistung auf einem hohen Niveau liegt (Abbildung 6.3, Abbildung 6.4). Diese Differenz kann auf die beiden Controller (Sequencer und DSC) zurückgeführt werden, deren Latenzzeiten die Leistung verringern (Algorithmen 6.1 und 6.2). Obwohl beim Entwurf der Controller der optimierte Datenfluß der SAND/1-Architektur berücksichtigt wurde, setzten in erster Linie technische Gegebenheiten zahlreiche Grenzen. Aus Gründen der Flexibilität und der Kosten wurden der Sequencer und der DSC in FPGAs (Altera) implementiert. Hierbei ergaben sich unter anderem folgende Einschränkungen:

- Da die Anzahl verfügbarer PINs bei beiden Controllern nicht ausreichte, mußte die Übertragung von Macro-Commands zwischen DSC und Sequencer anstelle einer 32 Bit breiten Verbindung über einen 8 Bit breiten Bus realisiert werden. Dadurch dauert die Übertragung eines Macro-Befehls vier Takte anstatt nur einem Takt. (Jeder Schritt - siehe Abbildung 6.17 - benötigt einen Macro-Befehl).
- Die Komplexität einiger Zustandsautomaten erforderte eine Clock mit halber Taktfrequenz (25 MHz)

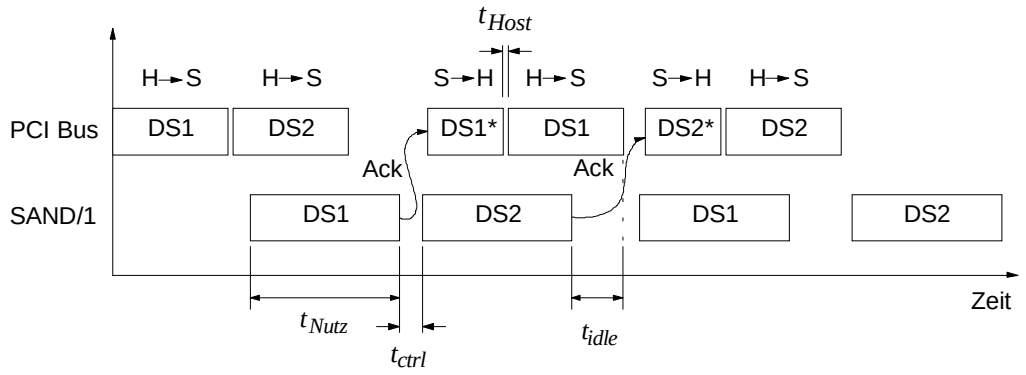
Darüber hinaus mußte aus Gründen der Komplexität auf die vollständige Ausnutzung der Überlappung beim Datentransport verzichtet werden. Anstatt das oben beschriebene Kriterium ($n \geq 4$) abzuprüfen, wurde die Übertragung der Ergebnisse so organisiert, daß mit Sicherheit keine Überschneidungen möglich sind. Die Summe dieser Punkte führt zu einem Leistungsverlust, der insbesondere bei kleinen Netzen deutlich wird. Die Latenzzeiten der Controller verlieren mit längeren Akkumulationszyklen an Einfluß, infolgedessen sich die tatsächliche Leistung der theoretisch erzielbaren annähert (Abbildung 6.4).

Ankopplung an den Host-Rechner

Der Vergleich der Leistung auf der PCI-Karte mit der im PC zur Verfügung stehenden Leistung zeigt, daß durch die Kopplung über den PCI eine erhebliche Leistungsminderung verbunden ist. Die in Kapitel 6.1.2 als Nutzungsgrad bzw. Wirkungsgrad ermittelte Größe verdeutlicht, daß insbesondere bei kleinen Netzen erhebliche Einbußen zu verbuchen sind (Nutzungsgrad < 10%), während bei großen Netzen bis zu 80% der Leistung im PC genutzt werden kann (Abbildung 6.4).

Die Ursache für diesen Leistungsverlust liegt in der Kommunikation zwischen der SAND/1-PCI-Karte und der Host-CPU. Während auf der SAND/1-PCI-Karte eine vollständig taktsynchrone Kommunikation vorliegt, erfolgt der Informationsaustausch mit dem Host über den PCI-Bus in asynchroner Form; zur Synchronisation beider Partner werden die beiden FIFOs (FIFO_IN, FIFO_OUT) eingesetzt. Der verwendete PCI Controller (AMCC S5933) erlaubt unter Benutzung sogenannter Mailbox-Register die Implementierung eines einfachen Protokolls, welches die Synchronisation steuert.

Um die überlappende Verarbeitung (siehe Kapitel 5) zu nutzen, ist es erforderlich, stets mit zwei Datensätzen zu operieren, wobei ein Datensatz aufgrund der systolischen Arbeitsweise von SAND/1 aus vier Mustern besteht. Während der erste Datensatz (DS1) in Bearbeitung ist, kann ein zweiter (DS2) vom Host an die SAND/1-PCI-Karte übertragen werden, und die Ergebnisse eines zuvor bearbeiteten ersten Datensatzes (DS1*) können an den Host zurück übertragen werden. In Abbildung 6.18 ist ein vereinfachtes Schema dieser asynchronen Übertragung dargestellt.



DS1/2 : Datensatz 1/2
DS1/2* : Ergebnis 1/2 basierend auf DS1/2
H→S : Übertragung vom Host zu SAND/1
S→H : Übertragung von SAND/1 zum Host
Ack : Acknowledge

Abbildung 6.18: Schematische Darstellung der zeitlichen Verhältnisse bei der Kommunikation über den PCI-Bus

Im Idealfall ist die SAND/1-Hardware ohne Wartezyklen ($t_{idle} \equiv 0$) mit der Verarbeitung zweier Datensätze DS1 und DS2 ausgelastet. Untersuchungen am realen System haben aber gezeigt, daß dies nicht der Fall ist. Hierfür sind hauptsächlich drei Gründe anzuführen:

- Die tatsächliche Übertragungsrate auf dem Bus liegt je nach System deutlich unter dem theoretischen Maximalwert vom 132 MBytes/s. Messungen haben ergeben, daß 25-40 MBytes/s eine realistische Größe darstellen. Die Ursache ist darin zu suchen, daß der PCI-Bus von weiteren Komponenten (Grafikkarte, Festplattencontroller, Netzwerkkarte etc.) benutzt wird und ein Master, z.B. der Host, eine laufende Übertragung unterbrechen kann. Da Adressen und Daten auf den gleichen Signalleitungen übertragen werden, stellen die 132 MBytes/s lediglich einen Grenzwert bei der Übertragung unendlich vieler Nutzworte dar. Ferner verursacht die PCI-To-Host-Bridge des Hostrechners eine von der Anzahl zu übertragender Datenworte unabhängige Verzögerung.

- Zwischen zwei Datensätzen ergibt sich eine Verzögerung (t_{ctrl}), die auf das Steuerwerk (DSC) der SAND/1-PCI-Karte zurückzuführen ist. Zwischen dem Eintreffen des Startbefehls (**`load_and_calc`**) und dem tatsächlichen Beginn der Ausführung liegt eine Latenzzeit, die einerseits aus der Abarbeitung einiger Zustandsautomaten des DSC herrührt und andererseits ihre Ursache in der byteweisen Übertragung der 32 Bit Macro-Commands zwischen DSC und Sequencer hat.
- Auch die Verarbeitung auf dem Host erzeugt eine Verzögerung (t_{Host}). Das in Abbildung 6.18 eingezeichnete Quittierungssignal ACK kann entweder als Interrupt oder aber per Polling ausgewertet werden. In beiden Fällen beeinflussen die Komplexität des übergeordneten Programms, das Betriebssystem, die Hosthardware und der Prozessortakt die Größe dieser Verzögerung.

Die hier dargestellte Problematik soll anhand eines konkreten Beispiels quantitativ untersucht und bewertet werden.

Detaillierte Problemanalyse

Zur detaillierten Analyse der oben dargestellten Problematik wird die Berechnung eines 16-16-16-Netzes auf dem SAND/1-PCI-Board (vier Neuro-Chips) betrachtet (Abbildung 6.19).

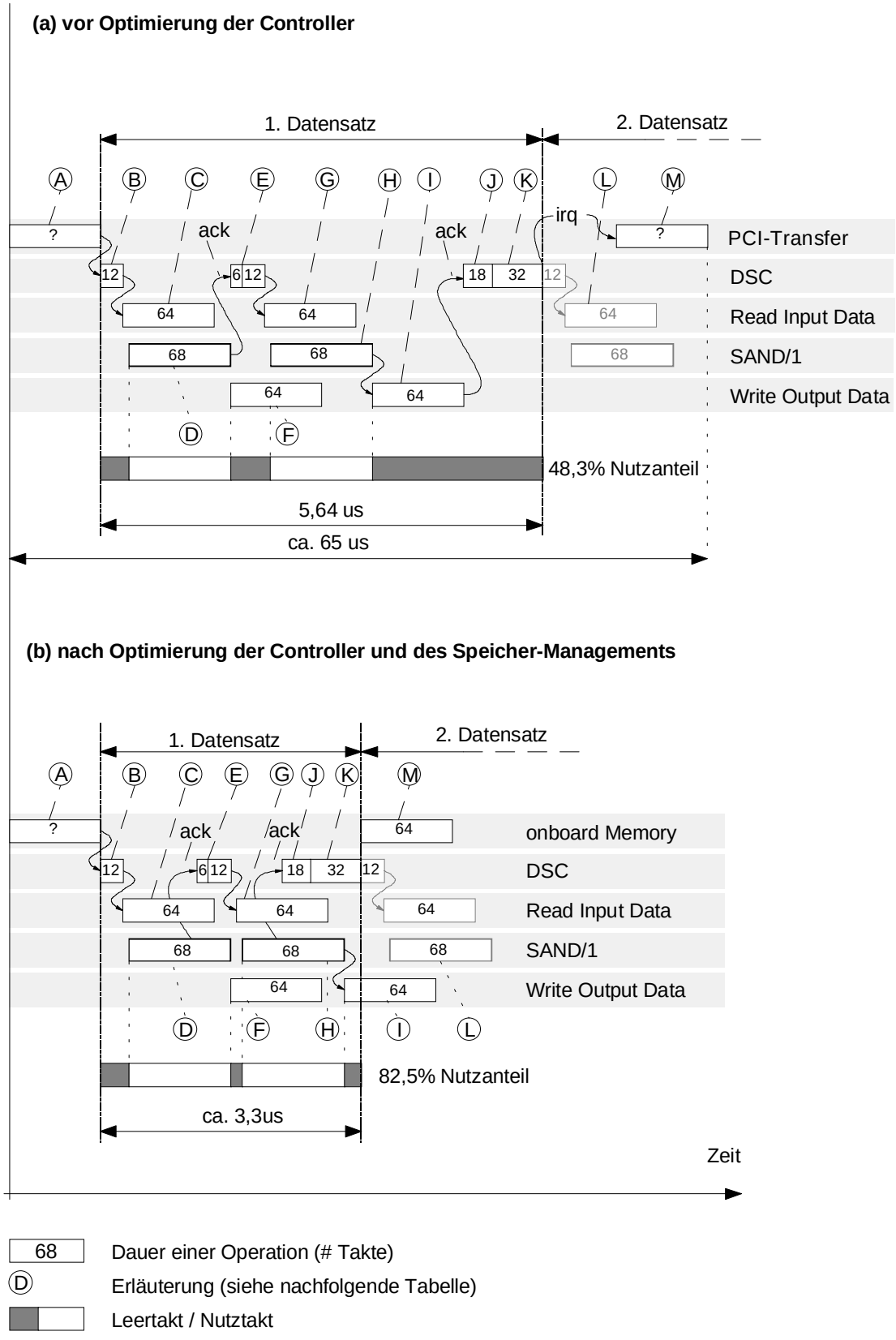


Abbildung 6.19: Berechnung eines 16-16-16-Netzes auf der SAND/1-PCI-Karte (vier Neuro-Chips) vor (a) und nach (b) der Optimierung von Controller und Speicher-Management

Vor der Optimierung der Controller setzt sich die Verarbeitung aus folgenden Blöcken zusammen (Abbildung 6.19 (a) / Tabelle 6.2):

Block	Dauer (#Takte)	Erläuterung
Ⓐ	?	Eingangsdaten (vier Muster zu je 16 Elementen) werden über den PCI-Bus an die SAND/1-Karte übertragen und dort im Eingangs-FIFO (FIFO_I) abgelegt. Die Dauer der Übertragung hängt vom Hostrechner ab und konnte mit den zur Verfügung stehenden Meßmitteln nicht isoliert ermittelt werden.
Ⓑ	12	Unmittelbar nach Abschluß der Übertragung (A) beginnt der Data-Stream-Controller (DSC), den ersten Macro-Befehl zur Berechnung der verdeckten Schicht an den Sequencer (SEQ) zu übertragen. Dieses 32-Bit-Befehlswort wird byteweise, unter Beachtung eines einfachen Protokolls, übertragen, so daß der gesamte Vorgang 12 Takte dauert.
Ⓒ	64	Unmittelbar nach Erhalt des Macro-Befehls beginnt der Sequencer, Daten aus dem FIFO_I an die vier parallelen SAND/1-Chips zu übertragen. Die Dauer entspricht exakt der Größe der Eingangsmatrix: 4 Muster zu je 16 Elementen ⇒ 64 Takte.
Ⓓ	68	Genau einen Takt nach dem Auslesebeginn (C) beginnen die vier SAND/1-Chips mit der Verarbeitung. Aufgrund der internen Pipeline-Struktur der SAND/1-Architektur dauert es 68 Takte, bis die Ergebnisdaten in den internen Registerbänken der SAND/1-Chips abgelegt sind.
Ⓔ	6+12	Unmittelbar nach Abschluß der Berechnung der verdeckten Schicht meldet der Sequencer mittels ACK dem DSC, daß er zur Ausführung des nächsten Befehls bereit ist. Zur Auswertung dieses ACK-Signals und zur Bereitstellung des nächsten Macro-Befehls (Ausgangsschicht) vergehen im DSC 6 Takte. Danach wird der zweite Macro-Befehl in der oben beschriebenen Weise an den SEQ übertragen, was wiederum 12 Takte erfordert.
Ⓕ	64	Parallel hierzu liest der SEQ die internen Registerbänke der vier SAND/1-Chips aus und speichert die Ergebnisse der verdeckten Schicht im FIFO_B ab.
Ⓖ	64	Nachdem der DSC den zweiten Macro-Befehl an den SEQ übertragen hat (E), kann dieser unmittelbar danach mit der Berechnung der Ausgangsschicht beginnen. Hierzu werden die Daten der verdeckten Schicht aus dem FIFO_B gelesen (der parallel noch beschrieben wird) und an die vier SAND/1-Chips übertragen.
Ⓗ	68	Die Berechnung der Ausgangsschicht in den vier parallelen SAND/1-Chips dauert wie schon zuvor 68 Takte.
Ⓘ	64	Der SEQ überträgt unmittelbar nach Abschluß der Berechnung die Ergebnisse in den FIFO_O. Allerdings wird das ACK-Signal an den DSC erst nach Abschluß dieses Speichervorgangs erzeugt.
Ⓙ Ⓚ	18+32	Der DSC wertet das ACK-Signal aus und erzeugt die für den PCI-Controller erforderlichen Signale (Mailbox2) (18 Takte), welcher darauf einen Interrupt auf der Seite des Hostrechners auslöst (32 Takte). Erst danach fährt der DSC mit der Verarbeitung weiterer Befehle fort.
Ⓛ	12+64	Sofern weitere Daten im FIFO_I bereitstehen, beginnt der DSC in der oben beschriebenen Weise mit deren Verarbeitung.
Ⓜ	?	Nachdem der Host-Rechner den Interrupt ausgewertet hat (IRQ), werden die Ergebnisse des ersten Datensatzes aus dem FIFO_O gelesen und über den PCI-Bus an den Host-Rechner übertragen. Wie schon beim Lesen der Eingangsdaten konnte auch hier mit den zur Verfügung stehenden Meßmitteln die Dauer dieser Übertragung nicht isoliert ermittelt werden.

Tabelle 6.2: Erläuterungen zur Abbildung 6.19

Die reine Verarbeitungszeit auf der SAND/1-PCI-Karte beträgt für einen Datensatz dieses Netzes (vier Muster zu je 16 Elementen) 5,64 μs. Hierin sind lediglich 2,7 μs Rechenzeit der SAND/1-Chips enthalten, der Rest setzt sich aus Transfer- und Latenzzeiten zusammen, so daß sich ein „Nutzanteil" von ca. 48,3% ergibt. Die gesamte Leistungsbilanz (gemessene Zeit im PC) weist eine Verarbeitungszeit von ca. 65 μs aus. Die erhebliche Differenz zwischen der Verarbeitungszeit auf der PCI-Karte und der im PC gemessenen Zeit rührt von einer Latenzzeit auf der Seite des Hostrechners her. Geht man von einer durchschnittlichen Übertragungsrate auf dem PCI-Bus von ca. 25 MBytes/s aus, dann dauert der Transfer von Ein- und Ausgangsdaten (256 Bytes) ca. 10,6 μs. Somit verbleibt ein Rest von ca. 50 μs. Untersuchungen mit verschiedenen Netzgrößen haben gezeigt, daß diese Latenzzeit von der Anzahl der übertragenen Datenworte über den PCI-Bus unabhängig ist.

Vor allem bei kleinen Netzen, deren Verarbeitungszeit sehr klein im Vergleich zur Latenzzeit (50 μs) ist, wirkt sich dieser Effekt fatal auf die gesamte Leistungsbilanz aus. Hinzu kommt noch das ungünstige Verhältnis von reiner Rechenzeit zu Transfer- und Latenzzeit auf der PCI-Karte (48,3% Nutzanteil), wodurch auch die Leistungsbilanz auf dem Board sehr ungünstig ausfällt.

Weitere Optimierungsschritte

Das oben dargestellte Beispiel verdeutlicht, daß es im wesentlichen zwei Ursachen für die nicht zufriedenstellende Leistung bei kleinen Netzen gibt:

- eine Latenzzeit von 50 μs bei der Übertragung von Daten über den PCI-Bus
- Latenzzeiten bei der Kommunikation der beiden Controller auf der PCI-Karte.

Um den Einfluß des erstgenannten Punktes zu reduzieren, müssen möglichst viele Datenblöcke (ein Datenblock entspricht vier Mustern) zusammenhängend übertragen werden. Dadurch erreicht man, daß der Anteil der Latenzzeit an der Gesamtverarbeitungszeit nur noch sehr gering ist. Insbesondere bei kleinen Netzen macht sich diese Maßnahme sehr günstig bemerkbar. Zur Realisierung muß das Speicherkonzept auf der PCI-Karte überarbeitet werden. Den Eingangs-FIFOs müssen (einige MBytes) große Speicher (SDRAMs) vorgeschaltet werden, die abwechselnd beschrieben und gelesen werden, um das Prinzip der überlappenden Verarbeitung (Abbildung 6.18) beibehalten zu können.

Die zweite Maßnahme betrifft die Latenzzeiten der Controller. Aus technischen Gründen ist es kaum möglich, diese Zeiten völlig zu eliminieren. Die Übertragung der Macro-Befehle muß auf der Ebene einzelner Bytes durchgeführt werden, da für eine 32-Bit breite Verbindung nicht genügend Pins zur Verfügung stehen. Durch eine geschicktere Abstimmung der beiden Controller kann aber erreicht werden, daß sich derartige Übertragungszeiten und die Verarbeitung von Daten auf den SAND/1-Chips überlappen. Hierfür muß in erster Linie das Handshake-Protokoll der beiden Controller so modifiziert werden, daß der SEQ bereits vor dem Beenden einer laufenden Operation das ACK-Signal an den DSC sendet, damit dessen Latenzzeit mit der Verarbeitung der Daten in den SAND/1-Chips überlappt. Durch die größeren Speicher auf dem Board kann gewährleistet werden, daß die Ergebnisse im FIFO_O sofort in den neuen Speicher ausgelagert werden können. Damit ist es nicht mehr erforderlich, daß der SEQ erst nach Abschluß der gesamten Operation das ACK-Signal an den DSC leitet, wodurch bei der bisherigen Lösung eine sehr lange Wartephase am Ende

einer Operation (Block I) entsteht. Diese Maßnahme hat zur Folge, daß sich die Verarbeitung ganzer Datenblöcke (vier Muster) wesentlich besser überlappt.

Unter Berücksichtigung dieser Maßnahmen ist das resultierende Zeitdiagramm für das oben betrachtete 16-16-16-Netz in der Abbildung 6.19 (b) dargestellt. Die Blöcke (A) bis (L) haben dabei die gleiche Bedeutung wie im oberen Teil des Bildes. Allerdings haben sich durch die Optimierungsmaßnahmen die zeitlichen Verhältnisse geändert, so daß der Nutzanteil in diesem Beispiel 82,5% betragen würde. Der mit (M) gekennzeichnete Block beschreibt nun nicht mehr die Übertragung der Ergebnisse über den PCI-Bus, sondern den Transfer der Daten vom FIFO_O in den neuen Speicher. Hierdurch ist es möglich, die nächste Operation schon mehr als 72 Takte früher zu beginnen, was sich insbesondere bei kleinen Netzen sehr positiv auswirken wird.

Die hier beschriebenen Schritte erfordern keine Modifikationen der SAND/1-Architektur sondern führen zu einer besseren Nutzung der parallelisierbaren Abläufe. Da für beide Controller eine VHDL-Beschreibung vorliegt, konnten diese Schritte bereits simuliert werden.

Um das Verhältnis von (langer) Übertragungszeit und (kurzer) Verarbeitungszeit zu verbessern, müßte ein schnellerer Bus, z.B. PCI66, eingesetzt werden [Shan95]. Auf der Seite des Hostrechners könnte die Verwendung schnellerer Speicher (100 MHz SDRAMs) für einen zügigeren Transport der Daten zur PCI-To-Host-Bridge sorgen. In der sogenannten PC-100-Spezifikation sind schnelle Speicher dieser Art vorgesehen [Intel98b].

SAND/1 versus leistungsfähige Standardprozessoren

Schon seit jeher ist die schnelle Ausführung der MAC-Operation, welche Basis zahlreicher Algorithmen der digitalen Signalverarbeitung ist, eine Domäne der DSPs. Durch die Möglichkeit, pro Takt zwei MAC-Operationen parallel auszuführen, scheint der DSP TMS320C6201 vor allem unter Berücksichtigung der hohen Taktfrequenz von 160 bzw. 200 MHz eine ernstzunehmende Alternative zu einem (Low-Cost) Neuro-Chip zu sein, da zumindest theoretisch eine Leistung von 320 bzw. 400 MMAC/s möglich ist.

Seitdem im PC-Bereich die Multimedia-Welle regelrecht boomt, hat die digitale Signal- und Bildverarbeitung auch in diesem Segment einen völlig neuen Stellenwert erhalten. Die Prozessorenhersteller, unter anderem auch INTEL, reagierten auf diese Entwicklung durch Erweiterung des bisher reinen SISD-Prozessor-Konzepts um eine SIMD-Komponente (MMX). Diesen Multimedia-Erweiterungen ist gemeinsam, daß sich Festkommaoperationen (z.B. MAC) ähnlich wie bei einem DSP sehr schnell (1 MAC/ Takt) und außerdem parallel (bis zu 4 16 x 16-Bit-Operationen) ausführen lassen. Vor allem unter dem Aspekt ständig steigender Taktfrequenzen läßt diese Technologie hohe Leistungen vermuten.

Die Untersuchungen in Kapitel 6.2 zeigen, daß die hohe theoretisch erreichbare Leistung beider Prozessoren (C6201 und INTEL Pentium II / MMX) nur in eingeschränktem Maße für die Beschleunigung neuronaler Operationen genutzt werden kann. Insbesondere beim C6201 wird deutlich, daß der parallele VLIW-Prozessorkern C6200 zwar recht hohe Leistungswerte erreichen kann (bis zu 270 MMAC/s Abbildung 6.12), diese aber in der Praxis deutlich geringer ausfallen.

Während der Datenfluß zwischen Registerbank und CPU-Kern so optimiert ist, daß pro Takt auch tatsächlich zwei MAC-Operationen ausgeführt werden können, verringern die Zugriffe auf externe Speicher die Leistung auf 7,5%.

Ganz ähnlich ist die Situation bei MMX - auch hier schränken die Zugriffe auf externe Speicher die Leistung ein. Das mehrstufige Cache-Konzept erfordert ein partielles Loop-Unrolling, für welches jedoch nicht genügend (schnelle) MMX-Register zur Verfügung stehen. Hieraus resultierende Ladezeiten verhindern, daß der maximale Durchsatz der Multiplizier-Pipeline (4 MAC-Operationen pro Takt bei drei Takten Latenzzeit) vollständig genutzt werden kann. Im Durchschnitt wird nur alle drei Takte ein PMADDWD-Befehl (4 MAC-Operationen) ausgeführt, so daß der Vorteil der Pipeline gar nicht genutzt wird. Eine weitere Einschränkung stellt die Breite der Akkumulationsregister von nur 32 Bit dar, was zur Folge hat, daß einer der beiden Operanden nur als Byte-Wert verarbeitet werden kann.

Bei beiden betrachteten Architekturen ist in erster Linie die unzureichende Abstimmung zwischen Durchsatz der CPU und dem Speicher Ursache der unbefriedigenden Leistungsbilanz. Zum Vergleich sind in der folgenden Abbildung die drei Architekturen mit den jeweiligen Bandbreiten einander gegenübergestellt. Anzumerken ist, daß es sich bei den angegebenen Werten um Maximalwerte handelt.

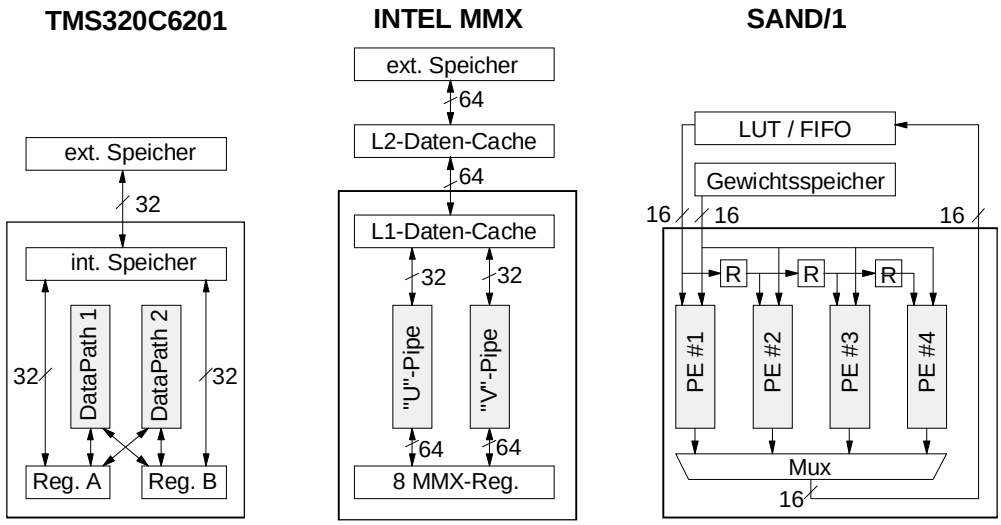


Abbildung 6.20: Ankopplung der drei Prozessoren TMS320C6201, INTEL Pentium II (MMX) und SAND/1 an externe Speicher

	TMS320C6201	INTEL MMX	SAND/1
Taktfrequenz (CPU)	160 MHz	266 MHz	50 MHz
Taktfrequenz (MEM)	80 MHz	66 MHz	50 MHz
Busbreite (MEM ⇒ CPU)	1 x 32 Bit (2 x 16 Bit)	1 x 64 Bit (4 x 16 Bit)	2 x 16 Bit (5 x 16 Bit)*
Busbreite (CPU intern)	2 x 32 + 2 x 40 Bit (4 x 16 + 2 x 32 Bit)	2 x 64 Bit (8 x 16 Bit)	8 x 16 Bit (32 x 16 Bit)*
Durchsatz maximal (MEM ⇒ CPU)	320 MBytes/s	528 MBytes/s	200 MBytes/s (800 MBytes/s)*
Durchsatz maximal (CPU intern)	1280 MBytes/s ⇔ 320 MMAC/s	2128 MBytes /s ⇔ 532 MMAC/s	800 MBytes/s (3200 MBytes/s)* ⇔ 200 MMAC/s (⇔800 MMAC/s)*
Durchsatz tatsächlich (CPU intern)	< 80 MBytes /s ⇔ 20 MMAC/s	< 280 MBytes /s ⇔ 70 MMAC/s	<760 MBytes/s (< 3040 MBytes/s)* ⇔ 190 MMAC/s (⇔760 MMAC/s)*

Tabelle 6.3: Vergleich von TMS320C6201, Intel-MMX und SAND/1

* Werte in Klammern stehen für das gesamte SAND/1-PCI-Board. Leistungsangaben bei SAND/1 und C6201 beziehen sich auf das jeweilige PCI-Board.

Zusammenfassung

Die theoretisch hohe Leistung der beiden Standardprozessoren C6201 und INTEL Pentium II (MMX) kann durch eine unzureichende Ankopplung an externe Speicher nur zu einem geringen Teil genutzt werden. Dagegen zeigt die Analyse der SAND/1-Architektur einen tatsächlichen Durchsatz, der sehr dicht an den theoretischen Wert heranreicht (Abbildung 6.21). Anzumerken ist, daß sich diese Angaben auf die Maximalleistung beziehen. Insbesondere das Steuerwerk der SAND/1-Architektur weist noch einige Mängel auf, die sich vor allem bei kleinen Netzen sehr negativ auswirken. Dahingegen wurde der Datenfluß soweit optimiert, daß hier ein Maximum an Leistungsfähigkeit zur Verfügung steht. Abgesehen von der Optimierung der Controller läßt sich die Leistung nur durch eine stärkere Parallelisierung oder durch eine höhere Taktfrequenz steigern.

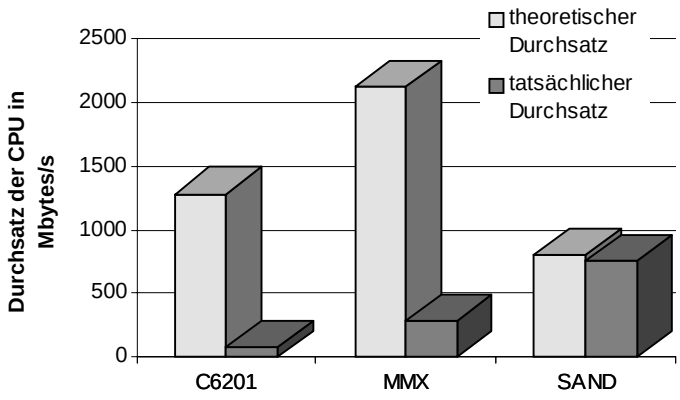


Abbildung 6.21: theoretischer und tatsächlicher Datendurchsatz der drei betrachteten Prozessoren

Obwohl beide Standardprozessoren mit wesentlich höheren Taktfrequenzen als SAND/1 arbeiten und beide in der Lage sind, Operationen zumindest teilweise parallel auszuführen, reichen die Leistungswerte bei weitem nicht an die von SAND/1 heran.

Die Leistung der beiden Standardprozessoren ließe sich leicht durch einen zusätzlichen Bus für die Gewichte bzw. Koeffizienten steigern. Bei der MMX-Architektur wäre ein zusätzlicher L1-Cache oder die Teilung des bestehenden L1-Caches in zwei Bänke mit jeweils 64 Bit Zugriff eine relativ leicht zu realisierende Erweiterung, die den tatsächlichen Datendurchsatz verbessern könnte. Trotzdem wird es auch mit dieser Maßnahme nicht möglich sein, die Leistung des Pentium-Prozessors in einem ähnlichen Maß wie bei SAND/1 zu nutzen. Eine Möglichkeit, diesen neuen Cache zu integrieren, ist in der Abbildung 6.22 dargestellt.

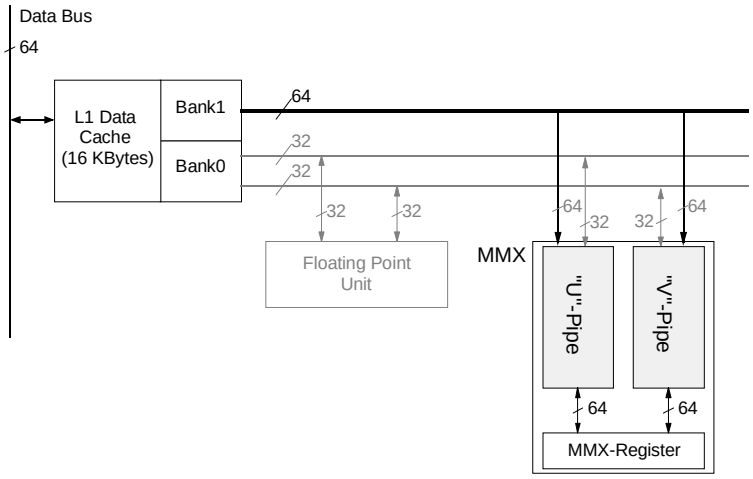


Abbildung 6.22: Erweiterung des Daten-Caches zur besseren Auslastung der MMX-Einheit

Mit der angekündigten Version C6201B von Texas Instruments wird durch die Unterteilung des internen Speichers in zwei Blöcke zu je vier Bänken ein simultanes Nachladen und Verarbeiten von Daten möglich. Nach wie vor bleibt aber auch bei dieser Architektur als Flaschenhals der 32-Bit-Bus zu externen Speichern bestehen. Hier könnte eine zusätzliche Verbindung mit den entsprechenden internen Speichermöglichkeiten für Abhilfe sorgen.

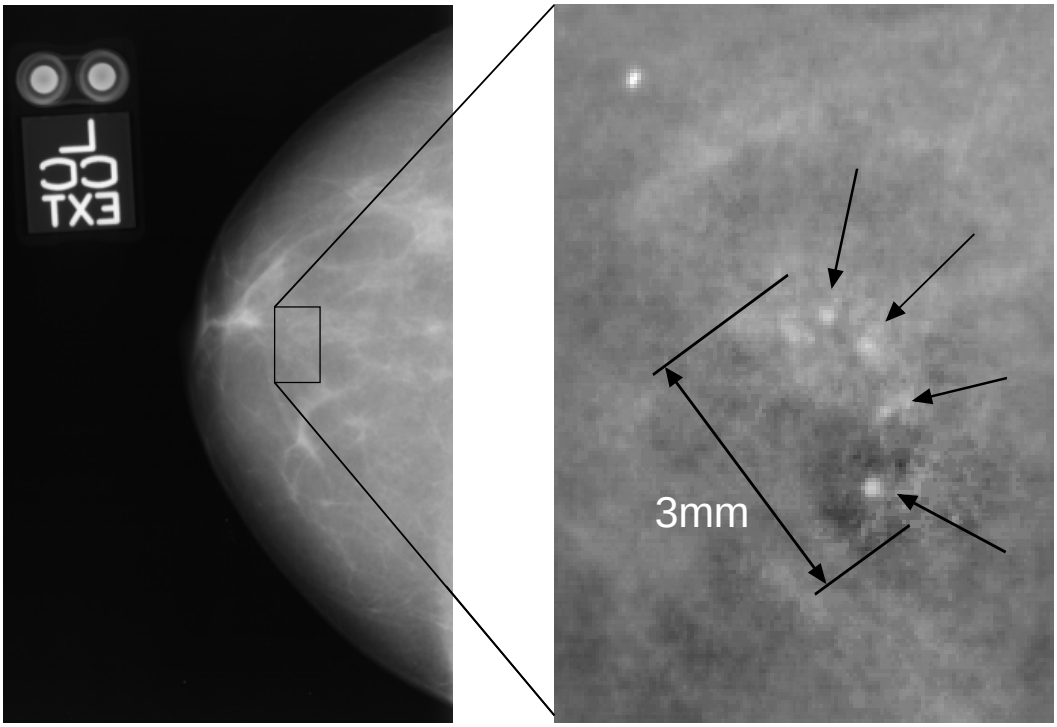
Beide vorgeschlagenen Maßnahmen laufen auf das gleiche Ziel hinaus. Für die Ausführung einer MAC-Operation pro Takt müssen gleichzeitig zwei Operanden geladen werden - nicht nur aus internen Registern, sondern auch von externen Speichern. Diese Forderung hat zur Folge, daß zwei separate Datenpfade (Aktivitäten / Gewichte) vorgesehen werden müssen. Im allgemeinen lassen sich die Gewichte so im Speicher anordnen, daß eine sequentielle Adreßgenerierung ausreichen würde - der Zusatzaufwand würde sich daher in überschaubaren Grenzen halten.

Aber auch dann ist der relativ langsame (50 MHz) SAND-Chip noch der nächsten Prozessorgeneration (500 - 1000 MHz) überlegen. Darüber hinaus ließe sich mit Hilfe der durchgehenden VHDL-Beschreibung die SAND-Architektur mit relativ geringem Aufwand in einer schnelleren Technologie implementieren, so daß sich der hier aufgezeigte Vorsprung gegenüber noch höher getakteten Prozessoren auch in Zukunft halten ließe.

7 Anwendung in der Medizintechnik

7.1 Mammographie

Die häufigste Krebserkrankung von Frauen in den westlichen Industrieländern ist der Brustkrebs. Allein in den USA ist im Jahre 1998 mit 178 700 neuen Erkrankungen und 43 500 Todesfällen zu rechnen [NCI98]. Eine deutliche Senkung der Sterberate ist durch eine frühzeitige Erkennung der Erkrankung im Rahmen regelmäßiger Vorsorgeuntersuchungen möglich. Hierfür hat sich die Röntgenmammographie etabliert, bei der die Brust unter zwei verschiedenen Einstrahlwinkeln (cranio-caudal, oblique oder medio-lateral) aufgenommen wird.



Digitalisiertes Mammogramm Bildausschnitt mit Mikroverkalkungen

Abbildung 7.1: Digitalisiertes Mammogramm (links) und Ausschnittsvergrößerung mit Mikroverkalkungen (rechts)

Auf einem Mammogramm (Abbildung 7.1, linker Teil) sind die anatomischen Bestandteile der Brust und eventuell Blutgefäße zu erkennen. Darüber hinaus findet man Merkmale, die auf einen pathologischen Befund hindeuten können. Hierzu zählen Gewebeverdichtungen und Kalkablagerungen, sogenannte Mikrokalzifikationen. Diese liefern vor allem dann einen Hinweis auf ein verdächtiges Gebiet, wenn sie in gruppierter Form auftreten (Abbildung 7.1, rechter Teil). Aufgrund ihrer geringen Größe (50 - 200 μm) können sie vom Radiologen leicht übersehen werden. Untersuchungen haben gezeigt, daß das Hinzuziehen eines zweiten Experten das Risiko einer Fehldiagnose deutlich reduzieren kann.

Die Aufgabe des zweiten Experten besteht im wesentlichen darin, den Radiologen auf alle verdächtigen Gebiete aufmerksam zu machen, um so die Gefahr einer Fehlklassifikation zu verringern. Mit den Mitteln der digitalen Bildverarbeitung läßt sich diese Aufgabe weitge-

hend automatisieren, so daß der zweite Experte mit Hilfe einer computergestützten Diagnosestation realisiert werden kann .

Im einzelnen sind folgende Schritte bei einer computergestützten Auswertung durchzuführen:

- **Digitalisierung der Röntgenbilder**. Mit einem Flachbettscanner werden die Röntgenbilder mit 50µm Auflösung und 12 Bit Grauwertdynamik digitalisiert. Übliche Bildgrößen sind ca. 4800 x 3600 Bildpunkte, so daß ein digitalisiertes Röntgenbild eine Größe von ca. 34,5 MBytes besitzt.
- **Detektion von Mikrokalzifikationen.** Die Detektion von Mikrokalzifikationen erfordert eine Bearbeitung des digitalisierten Bildes in mehreren Stufen. Mit Hilfe lateraler Filter (s. unten) werden Mikroverkalkungen verstärkt, so daß verdächtige Gebiete markiert werden können.
- **Dreidimensionale Rekonstruktion gruppierter Mikroverkalkungen.** Auf der Basis von zwei verschiedenen Ansichten einer Gruppe von Mikrokalzifikationen kann eine dreidimensionale Rekonstruktion durchgeführt werden [Müller98, Stotzka98], welche die diagnostische Entscheidung erleichtert.
- **Detektion von Gewebeverdichtungen.** Neben Mikrokalzifikationen sind auch Gewebeverdichtungen von pathologischem Interesse. Auch diese können mit entsprechend angepaßten lateralen Filtern in den Mammogrammen sichtbar gemacht werden.
- **Visualisierung.** Zuletzt müssen alle verdächtigen Gebiete, die segmentierten Mikroverkalkungen, ihre dreidimensionale Darstellung sowie die Gewebeverdichtungen auf einem hochauflösenden Bildschirm dargestellt werden.

Von allen hier aufgezählten Aufgaben ist die Verarbeitung der durchschnittlich 34,5 MBytes großen digitalisierten Mammogramme mit Hilfe lateraler Filter der rechenintensivste Schritt. Bei Maskengrößen von 3 x 3 bis 7 x 7 sind bis zu $1{,}2 \cdot 10^9$ Multiplikationen und Additionen erforderlich, deren Berechnung selbst auf einem schnellen Pentium II Prozessor (266 MHz) noch mehr als 8 Minuten dauert - eine Zeit, die im klinischen Alltag nicht akzeptabel wäre. Der Einsatz einer computergestützten Diagnosestation wird erst dann interessant, wenn die Auswertung innerhalb einiger zehn Sekunden möglich wird. Darüber hinaus dürfen auch die Kosten einer computergestützten Diagnose nicht übersehen werden. Eine Verringerung der Verarbeitungszeit läßt sich durch Parallelisierung erzielen. Eine Möglichkeit besteht darin, die Filteroperationen durch ein neuronales Netz nachzubilden, das dann mit Hilfe des SAND/1-Neuro-Chips parallel berechnet werden kann [Eppler98, Gemmeke98].

7.2 Detektion von Mikroverkalkungen mit Hilfe lateraler Filter

Zur Detektion von Mikroverkalkungen hat sich die Kombination aus einem Binomial- und einem Mittelwertfilter als besonders geeignet erwiesen. Das Mittelwertfilter realisiert einen Tiefpaß, der scharfe Grauwertübergänge glättet, während das Binomialfilter scharfe Übergänge verstärkt. Die Differenzbildung beider Teilbilder läßt Kanten im Mammogramm deutlicher hervortreten, wobei zu beachten ist, daß die Größen der Filtermasken unmittelbaren Einfluß auf die Breite der detektierten Kanten haben. Durch eine geeignete Wahl der Maskengrößen ist es möglich, Mikroverkalkungen deutlich hervorzuheben, deren Segementierung sich anschließend mit Hilfe einer Schwellwertfilterung realisieren läßt. Am Beispiel des

in Abbildung 7.1 dargestellten Bildausschnitts ist in Abbildung 7.2 die Filterung zur Detektion von Mikroverkalkungen schematisch dargestellt.

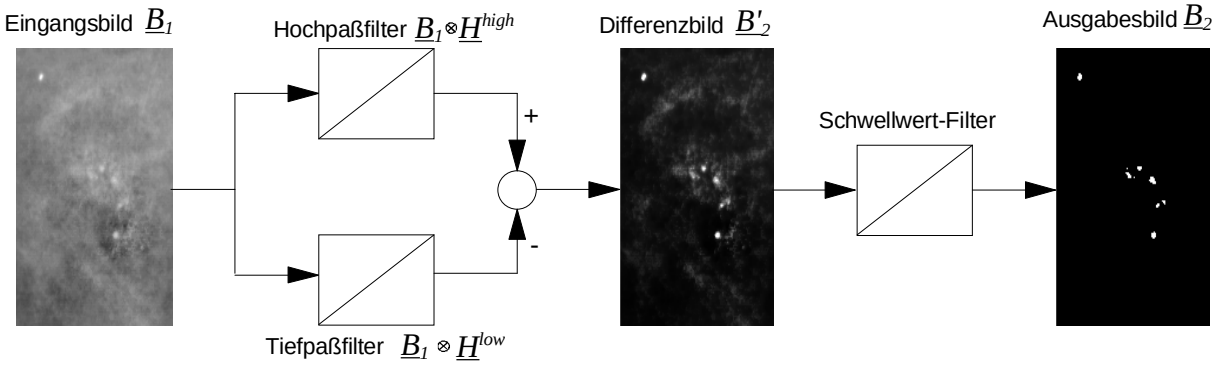


Abbildung 7.2: Kombination von Hochpaß-, Tiefpaß- und Schwellwertfilter zur Detektion von Mikroverkalkungen

Sowohl der Hochpaß als auch der Tiefpaß stellen eine lineare Abbildung in Form einer Faltung (4.12), (4.13) dar, die es erlaubt, beide Operationen in Gestalt einer Linearkombination zu einem Filter zusammenzufassen. Aus der Maske des Hochpaßfilters $\underline{H}^{high}(5x5)$ und der des Tiefpaßfilters $\underline{H}^{low}(7x7)$ entsteht hierdurch ein Bandpaßfilter (7.1), dessen Anwendung zum gleichen Ergebnis wie der parallele Einsatz des Hoch- und Tiefpaßfilters führt.

$$\underline{H}^{high}_{(5x5)} = \frac{1}{h^h_\Sigma}\begin{pmatrix} h^h_{1,1} & \cdots & h^h_{1,5} \\ & \ddots & \\ \vdots & h^h_{j,i} & \vdots \\ & \ddots & \\ h^h_{5,1} & \cdots & h^h_{5,5} \end{pmatrix} \quad mit\ h^h_\Sigma = \sum_j \sum_i h^h_{j,i}$$

$$\underline{H}^{low}_{(7x7)} = \frac{1}{h^l_\Sigma}\begin{pmatrix} h^l_{1,1} & \cdots & h^l_{1,7} \\ & \ddots & \\ \vdots & h^l_{j,i} & \vdots \\ & \ddots & \\ h^l_{7,1} & \cdots & h^l_{7,7} \end{pmatrix} \quad mit\ h^l_\Sigma = \sum_j \sum_i h^l_{j,i}$$

$$\underline{H}^{band}_{(7x7)} = \frac{1}{h^{all}_\Sigma}\begin{pmatrix} -h^l_{1,1}\cdot h^h_\Sigma & & \cdots & & -h^l_{1,7}\cdot h^h_\Sigma \\ & -h^l_{2,2}\cdot h^h_\Sigma + h^h_{1,1}\cdot h^l_\Sigma & \cdots & -h^l_{2,6}\cdot h^h_\Sigma + h^h_{1,5} h^l_\Sigma & \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ & -h^l_{6,2}\cdot h^h_\Sigma + h^h_{5,1}\cdot h^l_\Sigma & \cdots & -h^l_{6,6}\cdot h^h_\Sigma + h^h_{5,5}\cdot h^l_\Sigma & \\ -h^l_{7,1}\cdot h^h_\Sigma & & \cdots & & -h^l_{7,7}\cdot h^h_\Sigma \end{pmatrix}$$

$$mit \quad h^{all}_\Sigma = h^l_\Sigma \cdot h^h_\Sigma \qquad (7.1)$$

Durch diese Vereinfachung läßt sich die Filterung zur Verstärkung von Mikroverkalkungen in Mammogrammen auf eine Faltung (7.2) reduzieren, welche den Ausgangspunkt für die folgenden Betrachtungen darstellt.

$$\underline{B}_2' = \underline{B}_1 \otimes \underline{H}^{band}_{(7x7)} \qquad (7.2)$$

7.3 Äquivalenz von neuronalem Netz und lateralem Filter

Die mathematische Verwandtschaft von Faltung und neuronalem Netz (Multilayer-Perzeptron) wurde bereits in Kapitel 4.2.2 bei der Analyse parallelisierbarer Algorithmen deutlich. Zur Erinnerung sei im folgenden nochmals kurz dargestellt, wie sich die zweidimensionale Faltung als neuronale Operation darstellen läßt. Hierzu wird zuerst aus der zweidimensionalen eine eindimensionale Schreibweise gewonnen (4.14), die dann einen unmittelbaren Vergleich mit den Operationen eines MLP-Netzes erlaubt. Beide besitzen in bezug auf die Schreibweise des verallgemeinerten Neurons die gleichen h- und g-Funktionen (4.1), (4.15) und unterscheiden sich lediglich in der f-Funktion, die ohnehin in weiten Grenzen frei wählbar ist. Während bei MLP-Netzen üblicherweise Funktionen mit sigmoider Charakteristik verwendet werden, erfordert die Faltung keine explizite Transferfunktion, so daß in diesem Fall die Identität ($f \equiv g$) gewählt wird. Anschaulich betrachtet wird der Grauwert eines zu filternden Bildpunktes als Eingangswert der zu berechnenden Schicht eines MLP0-Netzes mit linearer Aktivierungsfunktion aufgefaßt. Da SAND/1 vier bzw. das voll bestückte PCI-Board 16 Funktionen in der Art von (4.14) parallel ausführt, können ebenso viele Bildpunkte parallel berechnet werden. Im Gegensatz zur Berechnung vollständig verbundener MLP-Netze besitzt bei der hier betrachteten Filterung jeder Bildpunkt einen individuellen Eingaberaum, eine sogenannte lokale Umgebung, die mit der benachbarter Punkte nur teilweise überlappt. Faßt man alle Punkte des Urbildes $\underline{B}_1$ als Neuronen der Eingangsschicht und alle Punkte des gefilterten Bildes $\underline{B}_2$ als Neuronen der Ausgangsschicht auf, so ergibt sich infolge des oben beschriebenen Effekts ein unvollständig verbundenes MLP0-Netz. Obwohl die SAND/1-Architektur nur für vollvernetzte Strukturen entworfen wurde, lassen sich dennoch auch teilvernetzte Strukturen auf SAND/1 abbilden, indem fehlende Verbindungen durch Null-Gewichte ersetzt werden. Allerdings kann der Vorteil des geringeren Rechenaufwandes, den teilvernetzte Strukturen mit sich bringen, auf SAND/1 nicht genutzt werden. Dieser Sachverhalt soll durch Abbildung 7.3 verdeutlicht werden.

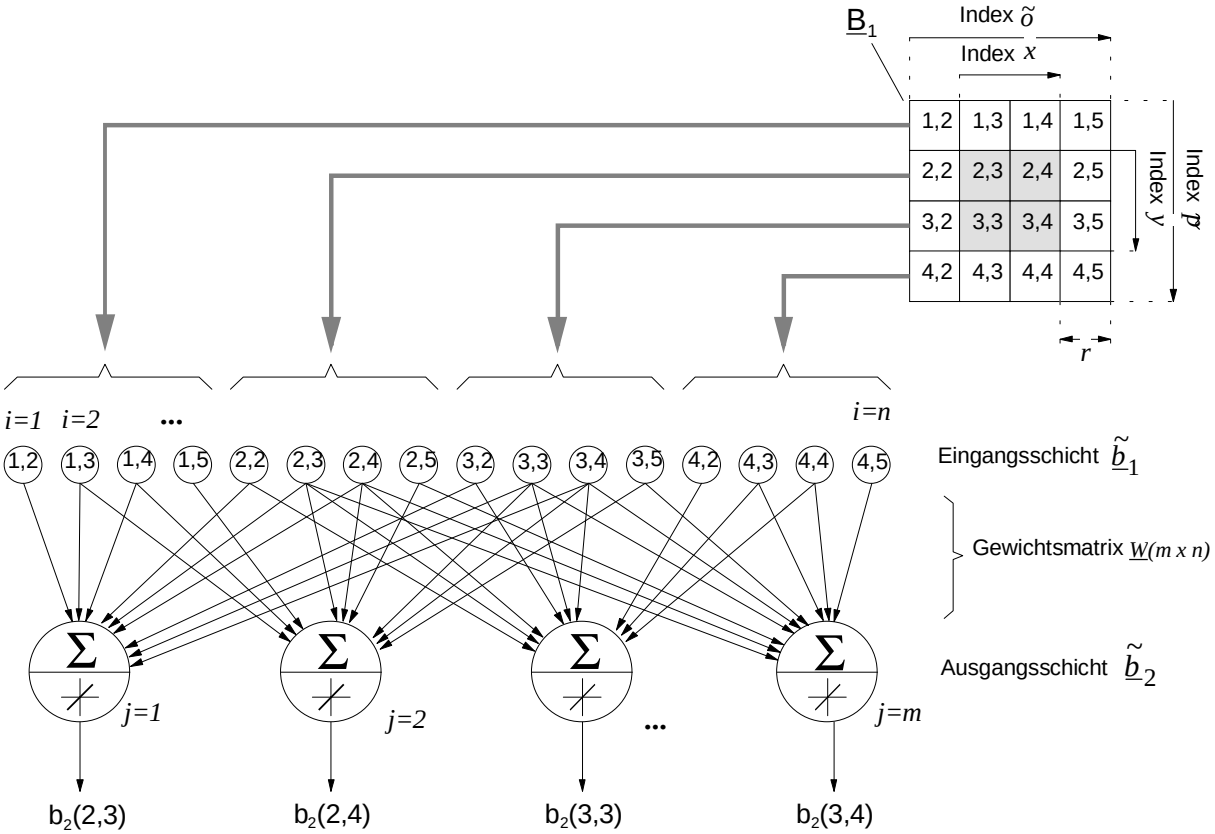


Abbildung 7.3: Zur Filterung (siehe Abbildung 7.3) äquivalentes Netz

Die Ausgabe dieses Netzes ist mathematisch gesehen völlig äquivalent zur der Ausgabe, die durch die Anwendung des Bandpaßfilters entsteht. Die Gewichtsmatrix dieses Netzes läßt sich unmittelbar aus der Filtermaske der Faltungsoperation ableiten. Ausschlaggebend sind die Anzahl der zu berechnenden Bildpunkte und die Größe der Filtermaske. Bei m zu berechnenden Bildpunkten (m parallele Filter), die in einem quadratischen Bildausschnitt angeordnet sind, und einer (quadratischen) Filtermaske der Seitenlänge ($2r$+1) ergibt sich die Anzahl der Eingangsneuronen (n) gemäß (7.3).

$$n = \left(\sqrt{m} + 2 \cdot r\right)^2 \qquad (\ 7.3\ )$$

Die Anzahl der Zeilen der Gewichtsmatrix (Anzahl der Ausgangsneuronen) entspricht der Anzahl der zu berechnenden Bildpunkte (m). Aus einer Filtermaske $\underline{H}$($2r$+1 x $2r$+1) läßt sich dann mit Hilfe des folgenden Algorithmus die Gewichtsmatrix $\underline{W}$(m x n) bestimmen. Die Bedeutung der Indizes und der Variablen sind der Abbildung 7.3 zu entnehmen.

Algorithmus 7.1

für $\tilde{o}$ *von 1 bis* $2r+\sqrt{m}$

  für $\tilde{p}$ *von 1 bis* $2r+\sqrt{m}$

    für $y=r+1$ *bis* $r-1+\sqrt{m}$

      für $x=r+1$ *bis* $r-1+\sqrt{m}$

        wenn $|\tilde{o}-x| \le r \wedge |\tilde{p}-y| \le r$ *dann* $\underline{W}(y,x) = h(\tilde{p},\tilde{o})$

        sonst $\underline{W}(y,x)=0{,}0$

      Ende x

    Ende y

  Ende $\tilde{p}$

Ende $\tilde{o}$

■

Erläuterung zum Algorithmus 7.1

Für jeden Punkt des betrachteten Bildausschnitts (Schleife $\tilde{o}$ und $\tilde{p}$) wird untersucht, ob dieser in den lokalen Umgebungen der zu berechnenden Bildpunkte liegt (Schleife x und y). Ist dies der Fall, dann wird an die entsprechende Stelle der Gewichtsmatrix der Koeffizient aus der Faltungsmaske geschrieben. Liegt der betrachtete Punkt außerhalb der lokalen Umgebung eines zu berechnenden Punktes, dann wird der Wert „0,0“ an der entsprechende Stelle der Gewichtsmatrix eingetragen.

Die folgende Tabelle zeigt die Berechnung der vier Bildpunkte des Beispiels aus Abbildung 7.3 für die Faltung und für das MLP0-Netz, zusammen mit der Anzahl erforderlicher MAC-Operationen.

Faltung		MLP0-Netz	
Berechnung	# MAC-Op.	Berechnung	#MAC-Op.
$b_2(2,3) = \sum_{o'=1}^{3}\sum_{p'=2}^{4} b_{1(o',p')} \cdot h_{(o',p')}$	9	$b_2(2,3) = \sum_{i=1}^{16} \tilde{b}_{1(i)} \cdot w_{(1,i)}$	16
$b_2(2,4) = \sum_{o'=1}^{3}\sum_{p'=3}^{5} b_{1(o',p')} \cdot h_{(o',p')}$	9	$b_2(2,4) = \sum_{i=1}^{16} \tilde{b}_{1(i)} \cdot w_{(2,i)}$	16
$b_2(3,3) = \sum_{o'=2}^{4}\sum_{p'=2}^{4} b_{1(o',p')} \cdot h_{(o',p')}$	9	$b_2(3,3) = \sum_{i=1}^{16} \tilde{b}_{1(i)} \cdot w_{(3,i)}$	16
$b_2(3,4) = \sum_{o'=2}^{4}\sum_{p'=3}^{5} b_{1(o',p')} \cdot h_{(o',p')}$	9	$b_2(3,4) = \sum_{i=1}^{16} \tilde{b}_{1(i)} \cdot w_{(4,i)}$	16

Tabelle 7.1: Gegenüberstellung von Faltung und neuronalem Netz

Während in diesem Beispiel die Faltung zur Berechnung eines Bildpunktes nur neun MAC-Operationen erfordert, benötigt das vollverbundene MLP0-Netz 16 MAC-Operationen pro Bildpunkt. Durch die Parallelisierung der SAND/1 Architektur lassen sich jedoch diese vier

Bildpunkte in 16 Takten berechnen, während die sequentiell ausgeführte Faltung 36 Takte erfordert.

7.4 Effizienz und Beschleunigung

Die Abbildung der zweidimensionalen Faltung auf ein MLP0-Netz hat, wie oben erläutert, zur Folge, daß Null-Gewichte ins Netz eingefügt werden müssen. Gegenüber der Berechnung mittels Faltung erhöht sich hierdurch die Anzahl der MAC-Operationen, so daß sich eine suboptimale Effizienz ergeben wird. Diese wird in erster Linie von der Anzahl parallel zu berechnender Bildpunkte und der Größe der Filtermaske beeinflußt. Im Hinblick auf den Einsatz des SAND/1-PCI-Boards mit maximal vier parallelen Neuro-Chips wurden die Beschleunigung gegenüber der sequentiellen Faltung und die Effizienz für Filtermasken der Größen (3 x 3), (5 x 5) und (7 x 7) untersucht. Betrachtet wurde die parallele Berechnung von 4, 9,16, 25 und 36 Bildpunkten. Bei der Betrachtung der Effizienz wurden immer nur so viele SAND/1 Chips berücksichtigt, wie zur Berechnung erforderlich sind, jedoch höchstens 16. Die Ergebnisse sind in der folgenden Abbildung zusammengefaßt.

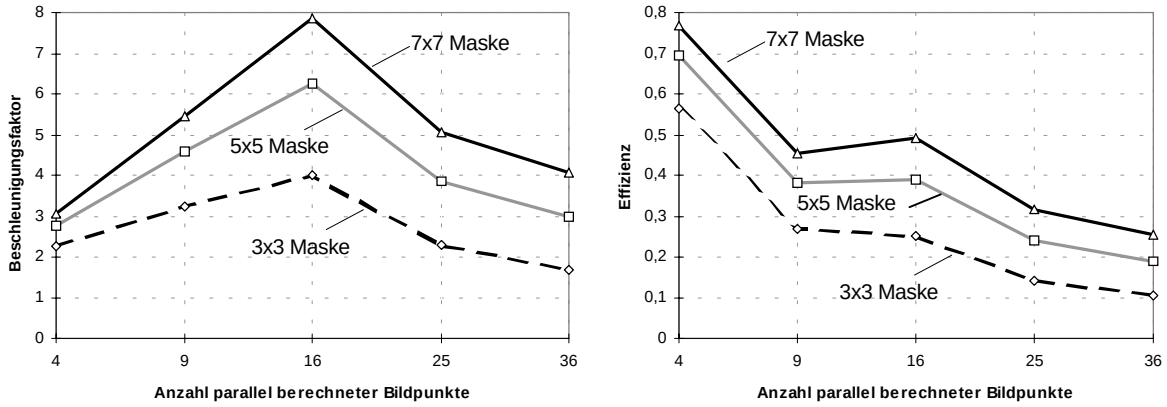


Abbildung 7.4: Beschleunigung und Effizienz für die Parallelisierung von Filteroperationen, aufgetragen über der Anzahl parallel zu verarbeitender Bildpunkte.

Generell führt eine große Faltungsmaske sowohl bei der Beschleunigung als auch bei der Effizienz zu besseren Ergebnissen. Dies liegt daran, daß die Größe der Filtermaske bei der parallelen Berechnung mit einem konstanten Faktor ins Gewicht fällt, während bei der sequentiellen Faltung dieser Faktor von der Anzahl der zu berechnenden Bildpunkte abhängt. Die höchste Effizienz erzielt man bei der parallelen Berechnung von lediglich vier Bildpunkten, welche den Einsatz nur eines SAND/1-Chips erfordert. Leider fällt hierbei die erreichbare Beschleunigung nur relativ gering aus ($b_4 = 3$). Stehen maximal vier SAND/1-Chips zu Verfügung (PCI-Board), dann scheint sich der Einsatz aller vier Bausteine zu lohnen, da bei der höchstmöglichen Beschleunigung ($b_{16} = b_{\max} = 8$) die Effizienz ($E_{16} = 0{,}5$) noch über dem Durchschnittswert der hier betrachteten Konfigurationen ($E_{Mittel} = 0{,}45$) liegt (7 x 7 Maske).

Mit Hilfe eines ersten Prototyps zur automatischen Detektion von Mikroverkalkungen in Mammogrammen, dessen Aufbau im folgenden Kapitel erläutert wird, wurde die tatsächlich erzielbare Leistung gemessen.

7.5 Einsatz in der Mammographie-Workstation

Mit dem ersten Prototyp der Mammographie-Workstation lassen sich Mammogramme digitalisieren und anschließend mit Hilfe der SAND/1-PCI-Karte auf gruppierte Mikroverkalkungen hin untersuchen. Ein vereinfachtes Schema dieses ersten Prototyps ist in Abbildung 7.5 dargestellt.

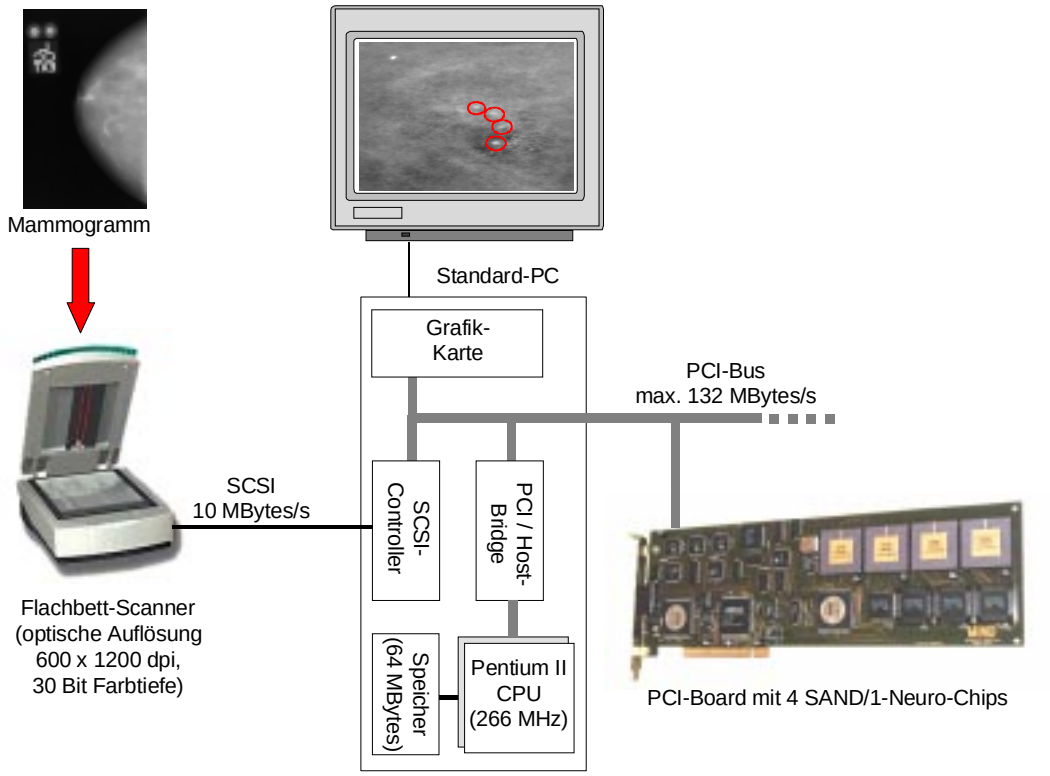


Abbildung 7.5: Schematischer Aufbau des ersten Prototyps einer Mammographie-Workstation zur Detektion von MIkroverkalkungen

Mit Hilfe eines Durchlicht-Flachbett-Scanners (600 x 1200 dpi optische Auflösung) werden Mammogramme digitalisiert und via SCSI im Hauptspeicher des PC's abgelegt. Eine für Windows95 erstellte Software mit grafischer Benutzeroberfläche erlaubt die Filterung digitalisierter Mammogramme zur Segmentierung von Mikrokalzifikationen. Wahlweise kann die Berechnung auf der Pentium-II-CPU oder auf der SAND/1-PCI-Karte durchgeführt werden. Die Messung der reinen Rechenzeit ermöglicht einen unmittelbaren Vergleich beider Varianten. Für diese Applikation wurde die C++-Bibliothek für das SAND/1-PCI-Board um eine zusätzliche Funktion, die den Bedürfnissen der Bildauswertung entgegenkommt, erweitert. Während bei der Berechnung neuronaler Netze im allgemeinen Ein- und Ausgangsaktivitäten in Fließkommadarstellung verarbeitet werden, sind die Grauwerte eines Mammogramms als vorzeichenlose 16-Bit-Festkommawerte gespeichert. Hierdurch kann die Datenaufbereitung stark vereinfacht werden, so daß die Daten bis auf wenige Modifikationen direkt vom Hauptspeicher des PC's an die SAND/1-PCI-Karte übertragen werden können.

Die Abbildung 7.6 zeigt einen Screenshot des Prototyps. Im oberen Bild (Original) ist der Ausschnitt eines Mammogramms mit einer Größe von ca. 850 x 1200 Punkten dargestellt. Die nur schwach sichtbaren Mikroverkalkungen wurden in der Abbildung mit einem weißen Kreis markiert. Im unteren Bild ist das Ergebnis der Filterung dargestellt. Die Mikroverkalkungen sind hier wesentlich deutlicher zu erkennen. Die Berechnung dieses Bildausschnitts dauert auf der SAND/1-PCI-Karte 0,42 Sekunden, auf der Pentium-II-CPU 6,4 Sekunden. Zur Visualisierung dieses Geschwindigkeitsunterschiedes werden im unteren Teil des Bildschirms zwei Balken (oben SAND/1, unten Pentium II) der entsprechenden Länge eingeblendet.

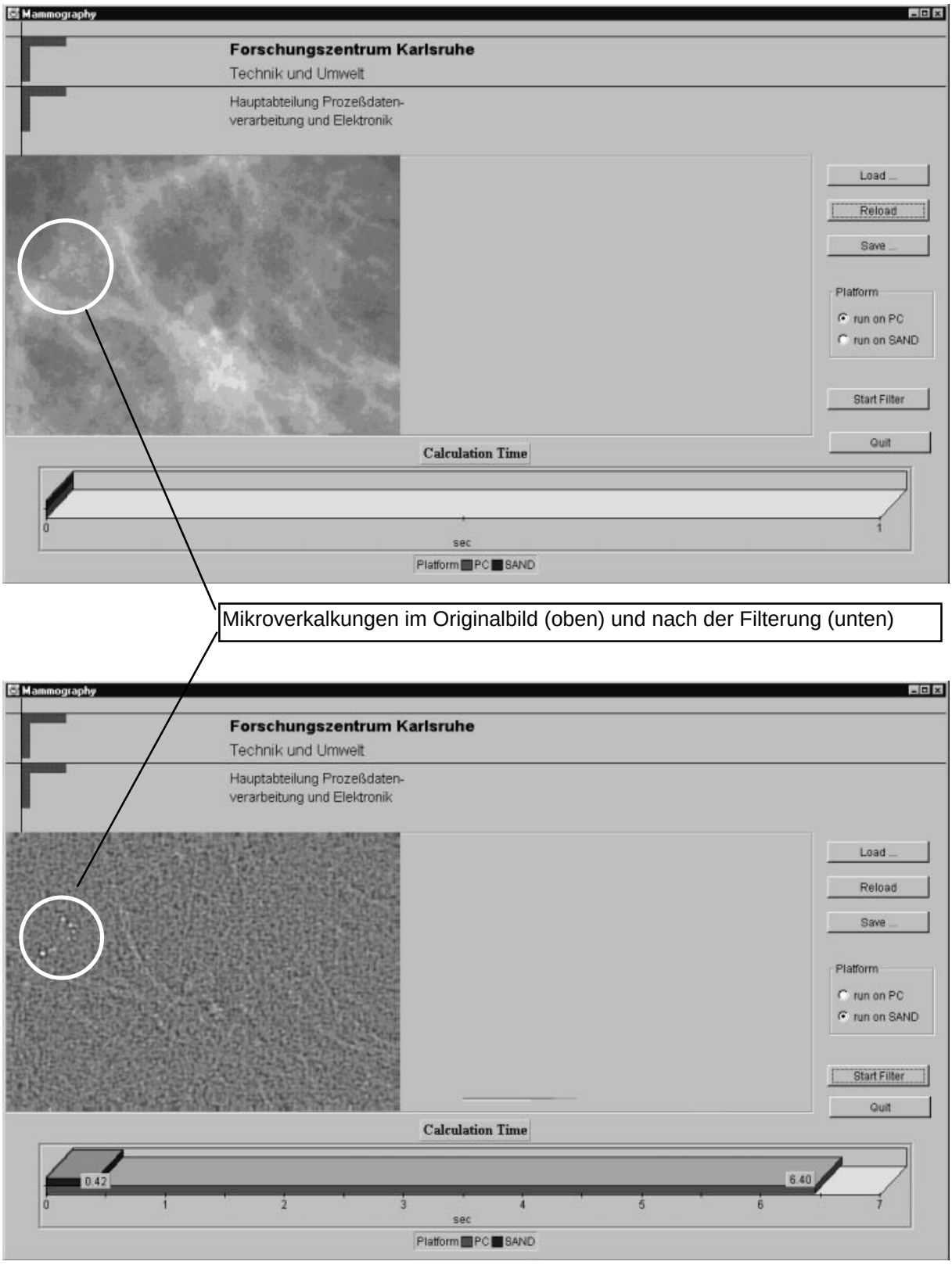


Abbildung 7.6: Filterung eines Ausschnitts eines Mammogramms mit der Mammographie-Workstation. Bild oben: Original, Bild unten: nach der Filterung. Rechenzeiten: SAND/1 (oberer Balken): 0,42s, Pentium II 266 MHz (unterer Balken): 6,4 s.

Mittlerweile wurde diese Filterfunktion in ein medizinisches Software-Paket in Form eines sogenannten „Plug-Ins“ integriert. Hierdurch ist es möglich, Patientendaten mit Hilfe einer

Datenbank zu verwalten und in derselben Software-Umgebung Funktionen zur Unterstützung der Diagnose auszuführen.

7.6 Ergebnisse

Die Vergleichsmessungen haben gezeigt, daß im Mittel die Filterung eines Mammogramms (4800 x 3600 Pixel) auf der SAND/1-PCI-Karte ca. 10 mal schneller als auf einer Pentium-II-CPU durchgeführt werden kann. Die hohe Verarbeitungsleistung der SAND/1-PCI-Karte und die hieraus resultierende kurze Filterzeit ermöglichen den Einsatz der Mammographie-Workstation im klinischen Alltag. Vor allem aber die Gesamtkosten von voraussichtlich 50.000.- DM liegen deutlich unter denen vergleichbarer Grafik-Workstations (ca. 200.000.- DM), welche für die Analyse von Röntgenbildern eingesetzt werden. Zu erwähnen sei an dieser Stelle noch die kurze Entwicklungszeit der Prototyp-Software. Mit einem Aufwand von lediglich drei Mann-Monaten wurde neben der grafischen Benutzeroberfläche (Java) die vollständige Anbindung der SAND/1-PCI-Karte durchgeführt (C/C++). In dieser kurzen Entwicklungszeit spiegelt sich die einfache Handhabung der parallelen Hardwarekomponente.

Trotz eines für die Praxis ausreichenden Beschleunigungsfaktors von 10 stellt sich die Frage, welche Komponente der Mammographie-Workstation verbessert werden könnte, um eine weitere Leistungssteigerung zu ermöglichen. Betrachtet man den Fluß der Daten von der Informationsquelle (Scanner) zur Senke (Grafikkarte), so werden zwei Schwachstellen deutlich:

- die Übertragung der Daten zwischen Scanner und Speicher erfolgt mit einer Geschwindigkeit von lediglich 10 MBytes pro Sekunde. Dieser Übertragungskanal stellt ohne Zweifel den Flaschenhals sämtlicher Übertragungswege der gesamtem Workstation dar;
- bevor die Daten mit Hilfe der SAND/1-PCI-Karte bearbeitet werden, erfolgt eine Speicherung im Memory des PCs. Dieser Zwischenschritt kostet sehr viel Zeit und ist aus Sicht der einzelnen Bearbeitungsschritte (scannen, filtern, visualisieren) nicht erforderlich.

Wesentlich günstiger scheint es, eine schnelle Verbindung zwischen der Akquisition der Daten und der SAND/1-PCI-Karte zu schaffen. Denkbar wäre daher, die SAND/1-PCI-Karte mit einer schnellen SCSI-Schnittstelle und entsprechend großem Speicher auszurüsten. Mit Hilfe dieser Optimierung wäre der Geschwindigkeitsvorteil durch die SAND/1-PCI-Karte noch deutlicher zu erkennen. Eine Erhöhung der Rechenleistung von SAND/1 selbst (Taktfrequenz, mehr ALUs pro Chip) scheint unter diesem Aspekt nur zweitrangig zu sein.

8 Zusammenfassung und Ausblick

Motiviert durch zahlreiche Anwendungen mit neuronalen Netzen im Forschungszentrum Karlsruhe (z.B. Triggerexperimente, Mammographie-Workstation), ergab sich die Notwendigkeit, neuronale Operationen durch eine geeignete Hardware zu beschleunigen. Die Analyse bereits am Markt etablierter Parallelprozessoren zur Beschleunigung neuronaler Netze verdeutlicht, daß diese ausnahmslos eine ganze Reihe von Problemen aufweisen, die einen Einsatz in der Praxis erheblich erschweren oder oftmals unmöglich machen. An erster Stelle sind hier eine viel zu komplexe Handhabung und ein untragbares Preis-Leistungsverhältnis zu erwähnen.

Ziel der vorliegenden Arbeit war es daher, eine Architektur zu entwickeln, die eine für den Anwender einfache Handhabung erlaubt und dabei die Besonderheiten einer besonders preisgünstigen Fertigungstechnologie (Sea-Of-Gates) berücksichtigt. Dabei sollte sich die Leistung von der heutiger Standardprozessoren deutlich abheben, um auch noch gegenüber der nächsten Prozessorgeneration einen Vorsprung behaupten zu können.

Anhand einer Auswertung von mehr als 150 Anwendungen mit neuronalen Netzen zeigte sich, daß für die Praxis vor allem Multilayer-Perzeptronen (MLP) und Kohonen-Netze eine wichtige Rolle spielen, so daß deren Implementierung in Hardware vorrangig betrachtet wurde. Durch die mathematische Verwandtschaft von Netzen mit radialer Aktivierungsfunktion (RBF) und MLP- bzw. Kohonen-Netzen wurden die RBF-Netze in die Betrachtungen mit einbezogen.

Beim Entwurf einer Architektur, die die oben formulierten Anforderungen erfüllt, wurden zwei aus der Literatur bekannte und bereits in verschiedenen Parallelprozessoren implementierte Architekturen herangezogen. Sogenannte One-To-All-Broadcast-Architekturen erlauben eine übersichtliche Programmierung, weisen aber den entscheidenden Nachteil auf, daß jedes der parallelen Prozeßelemente einen individuellen Gewichtsspeicher bzw. -bus benötigt. Somit scheiden diese Architekturen für eine kostengünstige Implementierung aus. Systolische Strukturen bieten hingegen durch eine überlappende Verarbeitung mehrerer Eingangsvektoren den Vorteil, daß lediglich ein Gewichtsbus für sämtliche parallelen Datenpfade genügt, wodurch sie prinzipiell für eine preiswerte Technologie (keine On-Chip-Speicher, wenige Busse zu externen Speichern) wesentlich besser geeignet wären. Sie haben aber den entscheidenden Nachteil, daß parallele Datenpfade untereinander Zwischenergebnisse austauschen, wodurch sich im allgemeinen nur auf der MAC-Operation basierende Algorithmen effizient in Hardware implementieren lassen. Das Beispiel des MA16 zeigt, daß zusätzlich zum systolischen Matrix-Matrix-Multiplizierer weitere Komponenten (Skalar-Multiplizierer, Matrix-Akkumulator) benötigt werden, um beispielsweise eine Abstandsberechnung (euklidische Distanz) durchführen zu können. Hierdurch erschwert sich die Handhabung enorm.

Aus beiden Architekturen wurde deshalb eine neue Architektur abgeleitet, die sich die Vorteile beider Varianten zunutze macht. So konnte erreicht werden, daß die Anzahl der Gewichtsbusse von der Anzahl der parallelen PEs unabhängig ist (nur ein Gewichtsbus) und gleichzeitig die PEs keine Zwischenergebnisse zu benachbarten PEs weiterreichen müssen, wie es sonst bei systolischen Architekturen notwendig ist. Durch diesen Trick ist es überhaupt erst möglich geworden, daß die Latenzzeit einer Operation (MAC, DISTL2) von

der Taktzeit der Kommunikationsstruktur weitgehend unabhängig wird. Dieser Punkt erhält insbesondere dann ein erhebliches Gewicht, wenn man beachtet, daß es bei einer Sea-Of-Gates-Implementierung unter Umständen kaum möglich ist, eine MAC- bzw. eine DISTL2-Operation innerhalb eines Taktes auszuführen, was bei einer rein systolischen Architektur zwingend erforderlich wäre. Als einzigen Nachteil dieser neuen Architektur ist zu erwähnen, daß die Latenzzeit eines Musters wie bei einem systolischen Feld von der Anzahl der parallelen Datenpfade abhängt. Dies ist darauf zurückzuführen, daß eine Matrix-Vektor-Operation durch eine Matrix-Matrix-Operation ersetzt wurde.

Durch die schichtweise Parallelisierung bei der Abbildung eines Netzes auf die neue Architektur ist es erforderlich, Ausgangsdaten der parallelen Rechenwerke wieder auf deren Eingänge zurückzukoppeln. In Verbindung mit der Festkommaarithmetik der Prozeßelemente ergibt sich hierdurch die Notwendigkeit, die Wortbreite am Ausgang der ALUs der Wortbreite der Eingänge anzupassen. Die anschließende Transformation der Ausgangswerte mit Hilfe einer bei neuronalen Netzen üblichen Transferfunktion mit sigmoidem Charakter verursacht ebenso wie die Anpassung der Wortbreite (Rundung) einen Fehler, der sich durch das Netz fortpflanzen würde. Ein im Rahmen dieser Arbeit entwickeltes Verfahren erlaubt die Minimierung dieser Fehler. Hierbei wird bei der Anpassung der Wortbreite eine Codierung eingefügt, die es erlaubt, den Dezimalpunkt so zu verschieben, daß bei der anschließenden Transformation auf der Basis einer sigmoiden Transferfunktion Quantisierungsfehler minimiert werden. Damit realisiert das Verfahren eine einfache Form der Fließkommaarithmetik, wobei die Elemente der Rechenwerke (Multiplizierer, Addierer) nach wie vor mit Festkommaarithmetik arbeiten.

Im Rahmen einer konkreten Realisierung wurden diejenigen Teile der neuen Architektur in Hardware implementiert, die für die Beschleunigung der Recall-Phase der betrachteten neuronalen Netze erforderlich sind. Für den hierbei entstandenen Neuro-Chip SAND/1 wurden eine PCI-Karte, die einen Parallelbetrieb von bis zu vier SAND/1-Chips erlaubt, und eine Treiberbibliothek entwickelt. Auf der Basis dieser Komponenten wurden zahlreiche Benchmarks durchgeführt, die grundsätzlich belegen, daß die Leistung der SAND/1-Hardware auf einem sehr hohen Niveau liegt. Auf der anderen Seite haben diese Untersuchungen auch einige Schwächen offenbart, die aber nur zu einem sehr geringen Teil durch die Architektur des Neuro-Chips SAND/1 begründet sind. Vielmehr sind die Kopplung der SAND/1-Hardware an einen Hostrechner über den PCI-Bus und die Controller auf der PCI-Karte für Leistungseinbußen vor allem bei kleinen Netzen verantwortlich. Im Rahmen dieser Arbeit wurde ein Vorschlag erarbeitet, der diese Schwachstelle weitgehend beseitigt. Hierfür müssen die Speicher auf der PCI-Karte vergrößert, das Speichermanagement angepaßt und die beiden Controller (DSC und Sequencer) optimiert werden.

Vergleichsmessungen mit schnellen Standardprozessoren wie dem Pentium II (MMX) oder dem DSP TMS320C6201 belegen, daß die SAND/1-Hardware einen deutlichen Vorsprung besitzt. Obwohl diese modernen Prozessoren mit sehr hohen Taktfrequenzen arbeiten, erreichen sie bei weitem nicht das Leistungsniveau der SAND/1-Hardware. Hierfür ist in erster Linie die Kopplung von Speicher und Prozessor verantwortlich. Während bei der SAND/1-Hardware das Speicher-Management so ausgelegt wurde, daß die hohe Leistung des Parallelprozessors optimal genutzt werden kann (eine Aktivität pro Takt), schränken bei den betrachteten Standardprozessoren die Zugriffe auf externe Speicher die Leistung erheblich ein. Vor allem bei großen Netzen wird dieser Nachteil deutlich, da die internen Speicher

(z.B. der Cache) für die Aufnahme der Matrizen zu klein sind. Darüber hinaus haben die Vergleichsmessungen gezeigt, daß ein weiterer Effekt die Leistung der Standardprozessoren mindert. Dieser Effekt macht sich hauptsächlich bei sehr kleinen Netzen bemerkbar. Während bei der MMX-Architektur eine konstante, von der Netzgröße unabhängige Latenzzeit die Verarbeitungszeit verlängert, bewirken beim DSP TMS320C6201 Loop-Prolog und -Epilog, daß die beiden parallelen Datenpfade nicht optimal genutzt werden können.

Die einfache Handhabung der SAND/1-Hardware gründet auf dem in dieser Arbeit entwikkelten neuen Kommunikationsschema. Es ermöglicht die Beschreibung sämtlicher Operationen mit Hilfe einer einheitlichen Parameterdarstellung, sogenannter Macro-Commands. Diese lassen sich automatisch aus einer Netzbeschreibung erzeugen, wodurch die Notwendigkeit einer hardwarenahen Programmierung des Parallelprozessors entfällt. Dadurch gestaltet sich der Einsatz der SAND/1-PCI-Karte so einfach, wie es ein Anwender sonst nur von einem Software-Werkzeug zur Simulation neuronaler Netze gewohnt ist: wenige Funktionsaufrufe genügen, um ein neuronales Netz mit der SAND/1-Hardware zu beschleunigen. Die hohe Benutzerfreundlichkeit ermöglichte eine rasche Anbindung zahlreicher Software-Werkzeuge (NeuroLution, Connect), welche mittlerweile zusammen mit der im Verlauf dieser Arbeit entwickelten Hardware kommerziell vertrieben werden.

Bei der Mammographie-Workstation des Forschungszentrums Karlsruhe wird die SAND/1-Hardware (Neuro-Chip und PCI-Karte) bereits erfolgreich zur Beschleunigung neuronaler Filter eingesetzt. Die dabei erzielte Beschleunigung (Faktor 10) gegenüber einem Pentium-II-Prozessor und die geringen Gesamtkosten der Mammographie-Workstation von weniger als 50.000.- DM ermöglichen den Einsatz im klinischen Alltag.

Die mittelfristigen Tätigkeiten in diesem Projekt werden die weitere Optimierung der SAND/1-PCI-Karte betreffen, um die Leistung des SAND/1-Neuro-Chips vor allem bei kleinen Netzen besser zu nutzen. Längerfristige Aufgabenstellungen ergeben sich unter Beachtung neuer Tendenzen im ASIC-Bereich. Insbesondere die Möglichkeiten von Systems-on-Chip eröffnen völlig neue Einsatzbereiche für neuronale Hardware. Die Entwicklung darf deshalb keinesfalls in der Single-Chip-Phase stecken bleiben. Komplexe Aufgabenstellungen in der Mustererkennung werden auch in Zukunft schnelle und vor allem parallele Rechenwerke erfordern. Wenngleich auch standardisierte Prozessoren wie DSPs sich mehr und mehr zu parallelen Architekturen hin entwickeln, wird aufgrund der notwendigen Flexibilität die Leistung deutlich hinter einer optimierten Architektur wie SAND/1 zurückbleiben. Die bisher gestellte Frage „Standardprozessor oder Spezialprozessor?“ wird an Bedeutung verlieren, da im Zuge von Systems-on-Chip die Kombination verschiedener Architekturelemente möglich wird. Die weiteren Arbeiten müssen sich daher mit einer geschickten Kopplung zwischen „Von-Neumann-Architekturen“ und spezialisierten Parallelarchitekturen befassen. Die im Rahmen dieser Arbeit entwickelte Architektur stellt eine interessante Ergänzung zu einer klassischen Architektur bei der Integration mehrere Komponenten für Systems-on-Chip dar, da sie einerseits leicht zu konfigurieren ist und andererseits keine besonderen Ansprüche an die Fertigungstechnologie stellt. Hierbei ist zu erwähnen, daß der Neuro-Chip SAND/1 und die beiden Controller (DSC und Sequencer) als VHDL-Modelle vorliegen, was den oben skizzierten Integrationsprozeß wesentlich erleichtert. Damit unterscheidet sich der SAND/1-Neuro-Chip in einem weiteren, für künftige Entwicklungen ganz wichtigen Punkt von allen anderen Neuro-Chips, für die es keine (synthesefähige) Verhaltensbeschreibung gibt.

Auch das Training neuronaler Netze sollte unter dem Aspekt einer höheren Integration diskutiert werden. In dieser Arbeit wurden bereits erste Untersuchungen durchgeführt (SAND/2). Anstelle zusätzlicher Rechenwerke, um z.B. das Backpropagation-Lernverfahren in Hardware zu implementieren, kann eine geschickte Kopplung eines DSP-Cores mit einem SAND-Core eine hohe Flexibilität bieten und hierdurch zahlreiche Einsatzmöglichkeiten im Bereich adaptiver Systeme unter Echtzeitbedingungen eröffnen.

Literaturverzeichnis

[Amd67] G. M. Amdahl: „Validity of the Single Processor Approach to Achieving Large Scale Computation Capabilities“ Proc. of AFIPS Spring Joint Conference, vol. 30, pp. 483-485 Atlantic City, NJ (USA), April 1967

[AMD98] Advanced Micro Devices Inc., „AMD-K6 MMX Enhanced Processor - Multimedia Technology“, http://www.amd.com/K6/k6docs/pdf/20726c.pdf, Oktober 1998

[Baker88] T. Baker, D. Hammerstrom: „Modifications to Artificial Neural Networks Models for Digital Hardware Implementation“, Technical Report No. CS/E 88-035, Department of Computer Science and Engineering; Oregon Graduate Center, 1988

[Becher96a] T. Becher: „Anwendung von Neuronalen Netzen zur Immisionsprognose in Luftgütemeßnetzen“, Tagungsband Anwendersymposium: Anwendung von Fuzzy Technologien und Neuronalen Netzen, S. 27-31, Berlin, 1996

[Becher96b] T. Becher, G. Hälsig: „Vorhersage hoher Ozonkonzentrationen mittels neuronaler Netze.“, Immissionsschutz (2) 1996, Erich Schmidt Verlag, Berlin, 1996.

[Becher97a] T. Becher, G. Kock, T. Fischer, W. Eppler, H. Gemmeke: „The MiND-project: building, applying and speeding-up neural networks using the SAND-neuroprocessor“, Proc. of the 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT97), pp. 2500-2503, 1997

[Becher97b] T. Becher, G. Kock, T. Fischer, W. Eppler, H. Gemmeke: „Einsteckkarte zur Entwicklung und Beschleunigung neuronaler Netze“, Maschinen Markt 45/97, S. 44-48, Vogel Verlag, 1997

[Beck90] J. Beck: „The Ring Array Processor (RAP): Hardware“, Technical Report *No. 90_048*, International Computer Science Institute, Berkeley, USA, 1990

[Boahen89] K. Boahen, P. Pouliquen, A. Andreou, R. Jenkins: „A Heteroassociative Memory Using Current-Mode MOS Analog VLSI Circuits“, IEEE Transactions on Circuits Systems vol 36, No. 5, pp 747 - 755, May 1989

[Brause95] R. Brause: „Neuronale Netze - Eine Einführung in die Neuroinformatik“, B.G. Teubner Stuttgart, 1995

[Brucke98] M. Brucke, W. Nebel, A. Schwarz, B. Mertsching, M. Hansen, B. Kollmeier: „Digital VLSI-Implementation of a Psychoacoustically and Physiologically Motivated Speech Preprocessor.“, S. Greenberg, M. Slaney, M. (eds.): Proc. of the NATO Advanced Study Institute on Computational Hearing, pp. 157-162, Il Ciocco, Italy, 1998

[Cabestany96] J. Cabestany, P. Ienne, J.M. Moreno, J. Madrenas: „Is there a Future for ANN Hardware?“ Proc. of the Workshop on Mixed Signal Design Integrated Circuits and Systems, Lodz, Poland, 1996

[Cern98] http://www1.cern.ch/NeuralNets/nnwInHepHard.html, August 1998

[CNAPS95] CNAPS Data Book 801-20063-03, Adaptive Solutions Inc., Beaverton, OR 97006, USA, March 1995

[Cruse97] C. Cruse, S. Leppelmann, A. Burwick, M. Bode: „Application of the Constructive Mikado-Algorithm on Remotely Sensed Data“, Neurocomputation in Remote Sensing Data Analysis, Springer Verlag, 1997

[CT97] c't Magazin für Computertechnik: „Großspurig: Ein kritischer Blick auf MMX“, Heft 1 S. 228 - 236, 1997,

[Dally90] J. W. Dally: „Performance Analysis of k-ary n-cube Interconnection Networks“, IEEE Transactions on Computers 39(6), pp. 775-785, June 1990

[dataf98] datafactory Informationssysteme: „neuroLution SAND/1 PCI Board Dokumentation Ver. 2.0“, Leipzig, 1998

[Duncan90] R. Duncan: „A Survey of Parallel Computer Architectures“, Computer 23 (2) pp. 5-16, February 1990

[Eppler96] W. Eppler, H. Gemmeke: „A New Fuzzy Controller Automatically Generated by a Neural Network", Proc. of Symposia on Intelligent Industrial Automation and Soft Computing, IIA'96 / SOCO'96, ICSC Acad. Press, GB, 1996

[Eppler97] W. Eppler, T. Fischer, H. Gemmeke, T. Becher, G. Kock: „High Speed neural Network Chip on PCI-Board“, Proc. of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems (MicroNeuro97), pp. 9-17, 1997

[Eppler98] W. Eppler, R. Stotzka, H.J. Neiber: „Schnelle Neuronale Filter in der Mammographie“, Jahrestagung der Deutschen Gesellschaft für Biomedizinische Technik e.V., S. 40-41, 1998

[Fischer96a] T. Fischer, W. Eppler, H. Gemmeke, A. Menchikov, S. Neußer: „Novel Digital Neural Hardware for Trigger Applications in Particle Physics“, Proc. of 2nd Workshop on Electronics for LHC Experiments, pp. 245-248, Balatonfüred, Hungary, 1996

[Fischer96b] T. Fischer, W. Eppler, H. Gemmeke: „Neuro-Chip SAND/1 für die einfache Handhabung in industriellen Anwendungen“, Tagungsband Anwendersymposium: Anwendungen von Fuzzy Technologien und Neuronalen Netzen, S. 119-125, Berlin, 1996

[Fischer97] T. Fischer, W. Eppler, H. Gemmeke, G. Kock, T. Becher: „The SAND Neurochip and its emebedding in the MiND System“, Proc. of the 7th International Conference on Artificial Neural Networks (ICANN97), pp. 1235-1240, 1997

[Fischer98] T. Fischer, W. Eppler, H. Gemmeke, A. Menchikov: „High Speed Neural Network Chip for Trigger Purposes in High Energy Physics“, Proc. on Design, Automation and Test in Europe Conference (DATE98*)*, pp. 108-115, 1998

[Flynn66] M.J. Flynn: „Very high speed computing systems“, Proc. IEEE 54 / 12, pp.1901-1909, 1966

[Flynn95] M.J. Flynn: „Computer Architecture - Pipelined and Parallel Processor Design“, Jones and Barlett Publishers London / Boston, 1995

[Friebe97] B. Friebe: „SIOP: Application-Specific Neural Hardware“, Proc. of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems (MicroNeuro97), pp.18-24, 1997

[Gemmeke96] H. Gemmeke, W. Eppler, T. Fischer, A. Menchikov, S. Neusser: „Neural Network Chips for Trigger Purposes in High Energy Physics“, Proc. of the Nuclear Science Symposium (NSS), pp. 302-306, Anaheim, USA, 1996

[Gemmeke97] H. Gemmeke, W. Eppler, T. Fischer, R. Stotzka, A. Chilingarian: „Performance of Neural Chip SAND/1 for Real Time Pattern Recognition“, Proc. of the 10th IEEE Real Time Conference (RT97), Beaune, France, 1997

[Gemmeke98] H. Gemmeke, W. Eppler, T. Fischer, R. Stotzka: „Der Neuro-Chip SAND und seine Anwendungen“, Wissenschaftliche Berichte FZKA 6080, 3. Statuskolloquium des Projektes Mikrosystemtechnik, S. 137-142, Forschungszentrum Karlsruhe, 1998

[Glesner94] M. Glesner, W. Pöchmüller: „Neurocomputers: An Overview of Neural Networks in VLSI“, Chapman & Hall, 1994

[Hammers90] D. Hammerstrom: „A VLSI Architecture for High-Performance, Low-Cost On-chip learning“, Proc. of International Joint Conference on Neural Networks, vol. 2, pp. 537-544, June 1990

[Hammers95] D. Hammerstrom : „A Digital VLSI Architecture for Real-World Applications“, An Introduction to Neural and Electronic Networks, 2nd Edition, Academic Press Inc., 1995

[Hebb49] D.O. Hebb: „The Organization of Behaviour“, Wiley, New York, Introduction and Chapter 4: The first stage of perception: growth of an assembly, pp. xi - xix, pp. 60-78, 1949

[Heemsk95] J. Heemskerk: „Overview of Neural Hardware“, Auszug aus der Dissertation „Neurocomputers for Brain Style Processing“, Universität Leiden (Niederlande), 1995

[Hennes90] L.H. Hennessy, D.A. Patterson: „Computer Architecture. A Quantitative Approach“, Morgan Kaufmann Publishers, 1990

[Holt93] J. Holt, J.-N. Hwang: „Finite Precision Error Analysis of Neural Network Hardware Implementations“, IEEE Transactions on Computers, Vol. 42, No. 3, pp. 281-290, 1993

[IBM98a] IBM Corp.: „ZISC (Zero Instruction Set Computer): User’s Manual“, IBM France, Component Development Laboratory, 1998

[IBM98b] http://www.ibm.fr/france/cdlab/cdlab.htm, Juli 1998

[Ienne93] P. Ienne, M. Viredaz: „GENES IV: A Bit-Serial Processing Element for a Multi-Model Neural Accelerator“, Proc. of the International Conference on Application-Specific Array-Processors, pp. 345-356, 1993

[Ienne94a] P. Ienne, M. Viredaz: „Bit-Serial Multipliers and Squarers“, IEEE Transactions on Computers, 43 (12), pp. 1445-1450, 1994

[Ienne95a] P. Ienne, G. Kuhn : „Digital Systems for Neural Networks“, Digital Signal Processing Technology, Society of Photo-Optical Instrumentation Engineers (SPIE) Vol. CR57, pp. 314-345, 1995

[IMS96] W. Beller, E. Bernrath, R. Hainbucher, T. Schwederski: „Gate Forest ® 0,8 µm CMOS Gate Array Cell Library“, Institut für Mikroelektronik Stuttgart (IMS), Januar 1996

[Intel98a] http://developer.intel.com/drg/mmx, Juli 1998

[Intel98b] http://developer.intel.com/design/pc98/, Oktober 1998

[Jabri96] M.A Jabri, R.J. Coggins, B.G. Flower: „Adaptive Analog VLSI Neural Systems“, Chaman & Hall, London, Glasgow, New York, pp. 57-88, 1996

[Jähne97] B. Jähne: „SIMD Bildverarbeitungsalgorithmen auf PC's, demonstriert am Multimedia Extensions- Instruktionssatz (MMX) von Intel“, Heidelberger Bildverarbeitungsforum, 1997

[Jones91] S. Jones, K. Sammut, Ch. Nielsen, J. Struanstrup: „Torodial Neural Network: Architecture and Processor Granularity Issues“, VLSI Design of Neural Networks, Kluwer Academic Publishers, pp. 229-254, 1991

[Keulen94] E. van Keulen, S. Colak, H. Withagen, H. Hegt: „Neural Network Hardware Performance Criteria“, Proc. of IEEE International Conference on Neural Networks, pp. 1885-1888, June 28 - July 1994

[Kock96] G. Kock, M. Endler, M. Gubitoso, S. Song: „Generating Parallel Code from High-Level Neural Network Descriptions“, Arbeitspapiere der GMD 1018, GMD - Forschungszentrum Informatik, 1996

[Koh82] T. Kohonen: „Self-Organized formation of topologically correct feature maps“, Biological Cybernetics 43, pp. 59-69, 1982

[Koll94] D. Koll: „Untersuchung effizienter Methoden der Parallelisierung neuronaler Netze auf SIMD-Rechnern“, Diplomarbeit Universität Karlsruhe (TH), Fakultät für Informatik, 1994

[Kumar94] V. Kumar, A. Grama, A. Gupta, G. Karypis: „Introduction to parallel Computing - design and anlysis of algorithms“, The Benjamin / Cummings Publishing Inc., 1994

[KungHT78] H.T. Kung, C.E. Leiserson: „Systolic Arrays for VLSI“ in Sparse Matrix Proc., pp. 256-282, Society of Industrial and Applied Mathematics, 1978

[KungHT80] H.T. Kung, C.E. Leiserson, „Algorithms for VLSI Processor Arrays“, chapter 8.3, Introduction to VLSI Systems, Addison-Wesley Reading, 1980

[KungSY89] S.Y. Kung, J.N. Hwang: „A unified systolic architecture for artificial neural networks“, Journal of Parallel and distributed Computing, Special Issue on Neural Networks, 6(2), April 1989

[KungSY91] S.Y. Kung: „Digital Neural Network Architecture and Implementation“, VLSI Design of Neural Networks, Kluwer Academic Publishers, pp. 205-227, 1991

[KungSY93] S.Y. Kung: „Digital Neural Networks“, Prentice Hall, New Jersey, pp. 340-368, 1993

[Lawrie75] Lawrie D.H.: „Access and alignment of data in an array processor“, IEEE Transactions on Computers C-24(1), pp. 1145-1155, 1975

[Leh93] C. Lehmann, M. Viredaz, F. Blayo: „A Generic Systolic Array Building Block for Neural Networks with On-Chip Learning“, IEEE Transactions on Neural Networks 4(3), pp. 400-407, Special issue on neural networks hardware, May 1993

[Lindsey94] C. Lindsey, T. Lindblad: „Review of Hardware Neural Networks: A User's Perspective“, Plenary Talk at the Third Workshop on Neural Networks*:* From Biology to High Energy Physics, Marciana Marina, Isola d'Elba, Italy, Sept. 26-30, 1994

[Lorenz97] E. Lorenz: „The MAGIC Telescope Project*",* Proc. of The Kruger National Park Workshop on TeV Gamma-Ray Astrophysics, pp. 23-30, South Africa, 1997

[Marwed92] P. Marwedel: „Synthese und Simulation von VLSI-Systemen - Algorithmen für den rechnerunterstützten Entwurf hochintegrierter Schaltungen“, Carl Hanser Verlag München und Wien, 1992

[Mauduit92] N. Mauduit, M. Duranton, J. Gobert, J.-A. Sirat: „Lneuro 1.0: A piece of hardware LEGO for building neural network systems“, IEEE Transactions on Neural Networks 3(3), pp. 414-422, 1992

[McCul43] W. McCulloch, W. Pitts: „A logical calculus of the ideas immanent in nervous activity“, Bulletin of Mathematical Biophysics 5, pp. 115-133, 1943

[Morgan90] N. Morgan: „The Ring Array Processor (RAP): Algorithms and Architecture“, Technical Report No. 90_047, International Computer Science Institute, Berkeley, USA, 1990

[Müller98] T. Müller, R. Stotzka, W. Eppler, H. Gemmeke: „Three-dimensional reconstruction of clustered microcalzifications“, Proc. of the 12th International Symposium and Exhibition on Computer Assisted Radiology and Surgery (CAR'98), p. 877, 1998

[MüllerU95] U. Müller, A. Gunzinger, W. Guggenbühl: „Fast Neural Net Simulation with a DSP Processor Array“, IEEE Transactions on Neural Networks, Vol. 6, No. 1, pp. 203-213, January 1995

[NCI98] National Cancer Institute: „Prevention of breast cancer“, Document No. 208/04730, 1998

[Neusser96] S. Neußer: „Parallel digital neural hardware for controller design“, Mathematics and Computers in Simulation 41, pp. 149-160, 1996

[Noll91] T. Noll: „Carry Save Architectures for High Speed Digital Signal Processing“, Journal of VLSI Signal Processing*,* vol. 3*,* pp. 121-140, 1990

[Nords92] T. Nordström, B. Svensson.: „Using and Designing Massively Parallel Computers for Artificial Neural Networks“, Journal of Parallel and Distributed Computing 14, pp. 260-285, 1992

[Ramach91] U. Ramacher, J. Beichter, W. Raab, J. Anlauf, N. Brühls, U. Hachmann, M. Wesseling: „Design of a 1st Generation Neurocomputer“, VLSI Design of Neural Networks, Kluwer Academic Publishers, pp. 271-310, 1991

[Ramach92] U. Ramacher: „SYNAPSE- A Neurocomputer That Synthesizes Neural Algorithms on a Parallel Systolic Engine“, Journal of Parallel and Distributed Computing, 14(3), pp. 306-318, Special issue on neural computing on massively parallel processing, 1992

[Rojas93] R. Rojas: „Theorie der neuronalen Netze: eine systematische Einführung“, Springer Verlag Berlin, Heidelberg, New York, 1993

[Rosenb58] F. Rosenblatt: „The perceptron: a probalistic model for information storage and organization in the brain“, Psychological review 65, pp. 386-408, 1958

[Roth96] B. Rothenberger: „Hardwareimplementierung des Backpropagation Lernverfahrens auf einem Neurochip“, Diplomarbeit an der Universität Karlsruhe (TH), Institut für Technik der Informationsverarbeitung, 1996

[Rum86] D.E. Rumelhart, G.E. Hinton, R.J. Williams: „Learning representations by back-propagating errors“, Nature 323, pp. 533-536, 1986

[Salmen95] „Massiv paralleles Rechnersystem CNAPS von Adaptive Solutions für die Echtzeitbildverarbeitung“, Applikationsschrift Cromemco GmbH, 1995

[Shan95] T. Shanley, D. Anderson: „PCI System Architecture“, Addison-Wesley Publishing Company, 1995

[Schlüß97] J. Schlüßler, I. Koren, J. Werner, J. Dohndorf, U. Ramacher, A. König: „Image Sensor with Analog Preprocessing“, Proc. of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems (MicroNeuro97), pp. 209-216, 1997

[Schrau98] N. Schraudolph: „A Fast, Compact Approximation of the Exponential Function“, Technical Report IDSIA-08-98, IDSIA, Cosro Elvezia 36, Lugano Switzerland, 1998

[Serb96] N. B. Serbedzija: „Simulating Artificial Neural Networks on Parallel Architectures“, Computer 29 (3) pp. 56-63, 1996

[Siegl90] J. Siegl, H. Eichele: „Hardwareentwicklung mit ASIC“, Einsatz und Anwendung von CAE Entwurfswerkzeugen, Hüthig Verlag Heidelberg, 1990

[Siemens92] Siemens AG: „Programmierbarer VLSI Array Prozessor für neuronale Netze und matrix-basierte Signalverarbeitung - Chiparchitektur, Entwicklungsumgebung, Applikationsbeispiele“, ZFE, 1992

[Siemens93] Siemens AG: „Beschreibung des MA16 für den Systementwickler - Architektur, Ansteuerung technische und elektrische Daten, Version 1.4“, ZFE, 1993

[Siemens94] Siemens AG: „SYNAPSE-1 - A General Purpose Neuro Computer“, ZFE, 1994

[Siggelkow91] A. Siggelkow, J. Nijhuis, S. Neußer, B. Spaanenburg: „Influence of Hardware Characteristics on the Performance of a Neural System“, Proc. of the 1991 Int. Conference on Artificial Neural Networks (ICANN91), Vol. 1, Finland, 1991

[Steinb61] K. Steinbuch: „Die Lernmatrix“, Kybernetik 1, S. 36-45, Springer Verlag Berlin, 1961

[Stotzka98] R. Stotzka, T. Müller, W. Eppler, H. Gemmeke: „Three Dimensional Reconstruction of Clustered Microcalzifications from two Digitized Mammogramms“, Proc. of SPIE’s International Symposium on Medical Imaging, p. 258, 1998

[Texas96] Texas Instruments Inc.: „Product Bulletin TMS320C8x DSPs“, Document No. SPRT112C, 1996

[Texas97a] Texas Instruments Inc.: „Technical Brief TMS320C62xx“, Document No. SPRU197a, 1997

[Trémiolles97] G. de Trémiolles, P. Tannhof, B. Plougonven, C. Demarigny, K. Madani: „Visual Probe Mark Inspection, using Hardware Implementation of Artificial Neural Networks, in VLSI Production“, Proc. of IWANN ’97, pp. 313-318, Lanzarote, Canary Islands, Spain, 1997

[Vincent91] J. Vincent: „Finite Wordlength, Integer Arithmetic Multilayer Perceptron Modeling for Hardware Realization“, in Delago-Frias et. al: VLSI for Neural Networks and Artificial Intelligence, pp.293-311, 1991

[Viredaz94] M. A. Viredaz: „Design and Analysis of a Systolic Array for Neural Computation“, École Polytechnique Fédérale de Lausanne (EPFL), Dissertation 1994

[Vixne96] P. Vixne, D. Formin, V. Chernikov: „A VLSI Digital Neural Processor with Variable Word Length of Operands“, Hard-Software systems for neural calculation support, http://www.module.vympel.msk.ru, Juli 1998

[Wahle94] S. Wahle : „Untersuchungen zur Parallelisierbarkeit von Backpropagation Netzwerken auf dem CNAPS Neurocomputer“, Diplomarbeit Universität Karlsruhe (TH), Fakultät für Informatik, Institut für Logik, Komplexität und Deduktionssysteme, 1994

[Walds96] K. Waldschmidt, S. Huss, M. Goedecke, A. Bleck: „Praktikum des modernen VLSI-Entwurfs“, B.G. Teubner Stuttgart, 1996

[Wawry93] J. Wawrzynek, K. Asanovic, N. Morgan: „The Design of a Neuro-Microprocessor“, IEEE Transactions on Neural Networks, 4(3) pp. 394-399, special issue on neural network hardware, May 1993

[Zell94] A. Zell : „Simulation Neuronaler Netze“, Addison-Wesley, 1994

[Zell95] A. Zell : „Simulation Neuronaler Netze auf MasPar, Paragon und Neurocomputer“, Benutzerinformation des Rechenzentrums Stuttgart, Heft1, 1995

Lebenslauf

Thomas Fischer

1. Oktober 1968	geboren in Stuttgart
1975 - 1979	Grundschule in Dornstetten
1979 - 1985	Gymnasium in Dornstetten
1985 - 1988	Eduard-Spranger-Wirtschaftsgymnasium in Freudenstadt Abschluß: allgemeine Hochschulreife
1988 - 1989	Wehrdienst
1989 - 1990	Praktika bei den Firmen ARBURG in Loßburg und MSC in Stutensee
1990 - 1995	Universität Stuttgart Studium der Elektrotechnik Abschluß: Dipl.-Ing.
1995 - 1998	Forschungszentrum Karlsruhe Doktorand in der Hauptabteilung für Prozeßdatenverarbeitung und Elektronik (HPE)
seit November 1998	Forschungszentrum Karlsruhe Nachwuchswissenschaftler in der Hauptabteilung für Prozeßdatenverarbeitung und Elektronik (HPE)