Abbildung komplexer, pulsierender, neuronaler Netzwerke auf spezielle Neuronale VLSI Hardware

Karsten Wendt, Matthias Ehrlich, Christian Mayr, René Schüffny

Inhalt— Im Rahmen des FACETS-Projektes ist die optimierte Abbildung neuronaler Netzwerke durch spezielle Algorithmen auf dafür konzipierte Hardware notwendig, um die Simulation plastischer und pulsierender Modelle zu ermöglichen. Die Erstellung der biologischen und Hardware-Modelle sowie die Konzeptionierung und Analyse der Algorithmen werden in dieser Arbeit vorgestellt.

Schlüsselwörter— Genetische Algorithmen, multikriterielle Optimierung, pulsierende neuronale Netzwerke, Parallelität

I. Einleitung

Die Simulation von großen neuronalen Netzen ist auf klassischen Rechnersystemen aufgrund der grundlegenden Strukturunterschiede nach wie vor problematisch, so dass im Rahmen des FACETS-Projekts eine speziell dafür konzipierte hochparallele VLSI-Hardware entworfen und umgesetzt wird. Die Übertragung von zu simulierenden Netzwerken in deren Konfigurationsspeicher kann als sehr komplexes multikriterielles Optimierungsproblem aufgefasst werden, was die Evaluierung und Implementierung geeigneter Algorithmen, wie z.B. multi-objektive genetische Algorithmen, in einem modularen, performanten und parallelrechnenden Framework nötig macht.

Es folgt zunächst die formale Analyse des zu optimierenden Problems (II), die Modellierung der biologischen und Hardware-Systeme (III) sowie die Vorstellung der umgesetzten Algorithmen (IV). Abschließend wird die Evaluierung der Verfahren erläutert und deren Ergebnisse zusammengefasst (V).

II. Problemanalyse

Die Anforderung des FACETS-Projektes an das *Mapping* (dt.: Abbilden) ist eine vollständige, verlustminimierte, zeitgerechte und optimierte Übertragungsvorschrift des biologischen Modells auf die Hardware. Formal lässt sich dieses Optimierungsproblem, bestehend aus der Abbildung der Neuronen und Synapsen sowie deren Parameter und Topologie auf entsprechende VLSI-Komponenten, als Abbildung:

$$m : H_p \rightarrow B_p \tag{1}$$

mit:

$$\begin{aligned} H_p &\subseteq H \\ B_p &\subseteq B \end{aligned} \tag{2}$$

beschreiben. H repräsentiert die Menge aller Hardware-Elemente und B die Menge aller biologischen Elemente. Demnach ergeben sich die Verlust- und Optimierungsfunktionen zu:

$$\begin{aligned} l_B &: B_p \rightarrow R,\ l_B(B/B_p) = 0 \\ c_B &: m \rightarrow R,\ c_B(m) \rightarrow \max. \end{aligned} \tag{3}$$

sowie:

$$\begin{aligned} l_H &: m \rightarrow R,\ l_H(m) = 0 \\ c_H &: m \rightarrow R,\ c_H(m) \rightarrow \max. \end{aligned} \tag{4}$$

l steht für den minimalen Verlust, d.h. keine nicht abbildbaren biologischen Komponenten oder unmögliche Hardware-anforderungen. c repräsentiert die optimale Umsetzung der Abbildung im Rahmen vorgegebener Beschränkungen, wie z.B. Parametergrenzen. Im Sinne einer Wichtung der einzelnen Kriterien wurde zur Vergleichbarkeit kein *Pareto*-Optimum angestrebt, sondern eine additive Reduktion auf einen skalaren Kostenwert gewählt .[1][2]

III. Modellierung

A. Graphendarstellung

Zur Systemmodellierung wird eine Graphendarstellung $G_{B,H}$ genutzt. Die Knoten V_G repräsentieren die Elemente und Parameter der Modelle, die gerichteten Kanten E_G die topologischen Eigenschaften sowie die semantische Zuordnung der Knoten zueinander.

Abbildung 1 zeigt beispielhaft die Modellierung eines neuronalen Netzwerks aus Neuronen (*N*) und Synapsen (*S*) als Graph mit Erhaltung der Topologie und Einführung spezieller Parameterknoten für die Parameter (*P*).

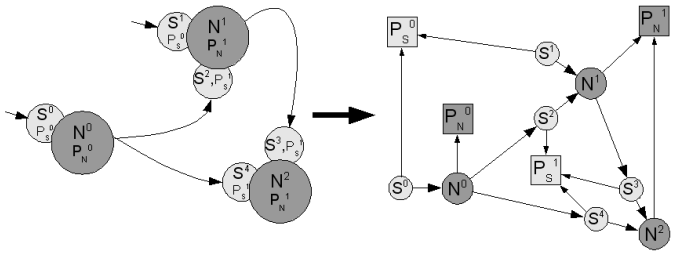

Abbildung 1: biologische Modellierung als Graph

Analog erfolgt die Umsetzung der Hardware-Modelle als Graph, was in Abbildung 2 beispielhaft dargestellt ist. Neuronen (*N*), verbunden mit einem synaptischen Feld (*S*),

Manuskript eingereicht am 12.03.2007. Die Arbeit wurde finanziert im Rahmen des *Information Society Technologies* Programms, Projekt FACETS. (Nr. 15879) Die Autoren sind am Stiftungslehrstuhls Hochparallele VLSI-Systeme und Neuromikroelektronik der TU Dresden tätig.

(Kontakt: Karsten Wendt, e-mail: wendt@iee.et.tu-dresden.de Telefon: +49 351 34591)

sind an ein Bussystem (*l1*), bestehend aus Kodierern (*WTA*) und Crossbars (*CB*), angeschlossen. Pulse der Neuronen können in diesem Beispiel über das Bussystem wieder in die Synapsen eingespeist werden. Die Modellierung erfolgt als Graph mit Erhaltung der Topologie und Einführung zusätzlicher Knoten für die Parameter (*P*).

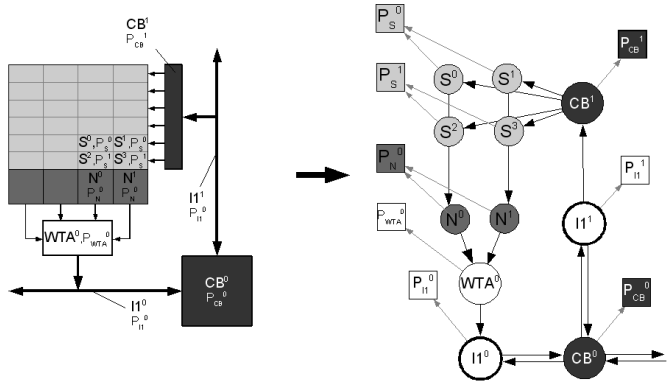

Abbildung 2: Hardware-Modellierung als Graph

B.Mapping als Graphenrelation

Resultierend daraus verallgemeinert sich die Abbildung *m* zu einer Graphenrelation r_m aus einer Menge von Tupeln

$$r_m \begin{cases} (b_0 h_i,..,h_k) \\ ... \\ (b_{\max} h_j,..,h_l) \end{cases} \quad (5)$$

mit

$$\begin{aligned} b_0,..,b_{\max} &\in V_B \\ h_0,..,h_{\max} &\in V_H \end{aligned} \quad (6)$$

V_B repräsentiert jeweils die Menge aller Knoten der biologischen und V_H die der Hardware-Graphenmodelle. Die Abbildung r_m ist nicht disjunkt, d.h. biologische Elemente können mehreren Hardwarekomponenten zugewiesen sein und umgekehrt.

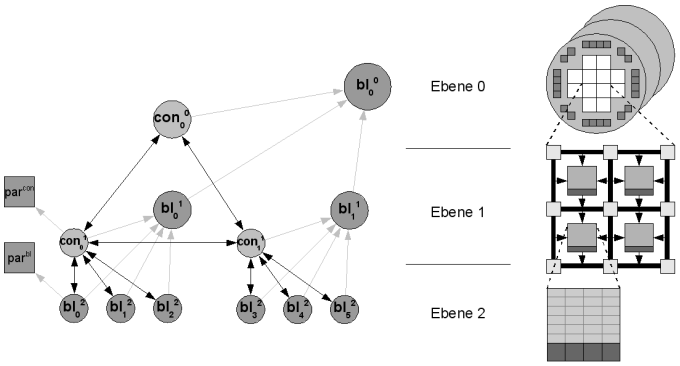

Abbildung 3: abstraktes hierarchisches Hardware-Modell

Zur Evaluierung der Algorithmen wird ein abstraktes hierarchisches Hardware-Modell, bestehend aus Hardwareblöcken für Neuronen und Kommunikationselementen für Synapsen, erstellt. Abbildung 3 zeigt ein schematisches Beispielmodell aus Blöcken $bl^{0..2}$ über drei Ebenen und zwei Kommunikationsschichten $con^{0,1.}$ Damit kann ein rechts angedeutetes FACETS-System aus mehreren Wafern auf der oberen Ebene sowie Neuronen- und Synapsenblöcken und deren Kommunikationselementen, bzw. die neuronalen und synaptischen Schaltungen selbst auf den unteren beiden Ebenen dargestellt werden.

Algorithmenintern werden die Modelle zur effizienten Verarbeitung in verschiedene Matrizen transformiert, welche die Topologie des Netzwerks, der Hardware sowie die Nutzungskosten und Beschränkungen der Hardwarekomponenten abbilden. Die Kostenfunktionen der Optimierung über r_m ergeben sich sinngemäß aus $l_{B,H}$ und $c_{B,H}$.

IV. Algorithmierung

A.Gesamter Mapping-Prozess

Der gesamte Mapping-Prozess ist in einzelne Mapping-Schritte unterteilt, was die Nutzung unterschiedlicher Optimierungsalgorithmen für jede Hardware-Ebene ermöglicht. Abbildung 4 zeigt an einem Beispiel den Ablauf des Mapping-Vorgangs. Beginnend auf der obersten Ebene wird das biologische Modell von einem Verfahren $NMappAlg_0^0$ in Teilnetze zerlegt und deren Neuronen den obersten Hardware-Knoten also z.B. verschiedenen Wafern zugeordnet. Untergeordnete Algorithmen wie z.B. $NMappAlg_1^1$ segmentieren die Teilnetze weiter und weisen sie entsprechenden Blöcken der nächsten Hardware-Ebene, d.h. verschiedenen Bereichen auf dem Wafer zu (siehe Abschnitt *B*).

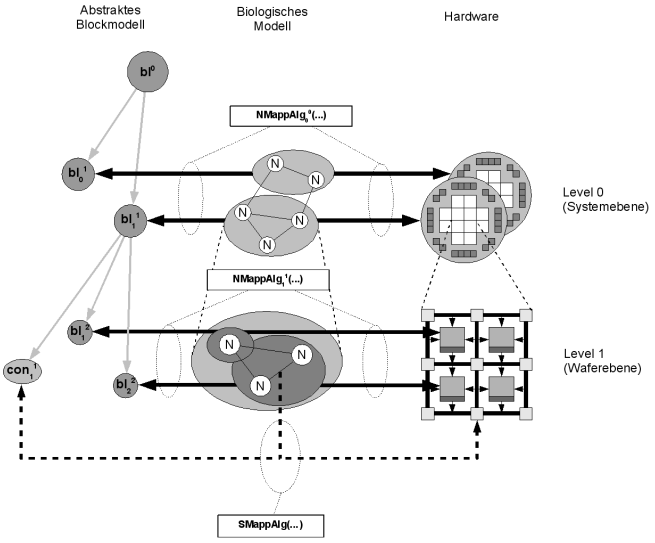

Abbildung 4: gesamter Mapping-Prozess

Nach der Abbildung aller Neuronen des biologischen Netzes ordnet abschließend ein Algorithmus *SMappAlg* die synaptischen Verbindungen den Kommunikationselementen des Hardware-Modells zu (siehe Abschnitt *C*). Dadurch verallgemeinern sich die Algorithmen zu Optimierungsverfahren für einzelne Hardware-Ebenen.

Die Entwicklung und Analyse der Algorithmen erfolgte in einem dafür entworfenen modularen C++ -Framework, was die Erweiterung der Algorithmen, Modelle und Kostenfunktionen unabhängig voneinander möglich macht.

B. Umgesetzte Algorithmen

Folgende Algorithmen wurden zur optimierten Abbildung der Neuronen auf die Hardware-Elemente konzipiert und umgesetzt. Aufgrund der in Entwicklung befindlichen Hardware des FACETS-Projekts konzentrieren sich die Verfahren auf die topologische Optimierung der Netze hinsichtlich der Hardwarebeschreibung.

1) Vollsuche

Die Vollsuche permutiert sequentiell die Neuronen-Abbildungen so, dass alle möglichen Zuordnungen mittels der Kostenfunktionen bewertet werden und so die beste Abbildung ermittelt wird. Aufgrund des *np*-schweren Problems wurden Optimierungen wie *Branch-And-Cut*-Meta-Heuristiken und angepasste Mapping-Reihenfolgen eingeführt. Der Algorithmus ist daher als exaktes Verfahren einzuordnen.[2][6]

2) Einsortierungsverfahren

Die Funktionsweise des Einsortierungsverfahrens beruht auf dem sequentiellen Abbilden noch nicht zugeordneter Neuronen zu den Hardware-Elementen so, dass jeweils ein minimaler Anstieg der Kosten erreicht wird. Zur Optimierung wurde eine angepasste Mapping-Reihenfolge eingeführt. Das Verfahren ist als *Greedy*- (dt.: gierig) Algorithmus einzustufen.[2]

3) Zufällig permutierende Suche

Die zufällig permutierende Suche wird mit einem randomisierten Initial-Mapping gestartet. Iterativ werden zwei Neuronenzuordnungen ausgewählt und permutiert. Werden dadurch die Kosten verringert, wird der Tausch beibehalten und ansonsten rückgängig gemacht. Das Verfahren terminiert nach einer gegeben Anzahl von Iterationen ohne Kostenverbesserung und ist als stochastische Suche einzustufen.[2]

Aufgrund der häufigen Berechnung des Kostengradienten bei einer Indexpermutation wurde ein spezielles Verfahren erstellt, was die effiziente Ermittlung des Gradienten bei den meisten Hardware-Modellen ermöglicht. Andernfalls ist die aufwändige Kostenevaluierung für das Gesamtsystem vor und nach der Permutation nötig. Abbildung 5 zeigt schematisch die Neuplatzierung der Synapsen (*schwarze Punkte*) in einer Verbindungsmatrix bei der Permutation der Neuronen x und y. Die Graufärbungen symbolisieren die verschiedenen Verbindungskosten zwischen den Hardwareblöcken b_0 bis b_5. So sind die veränderten Kosten der synaptischen Verbindungen zwischen Block b_0 und b_1 ersichtlich, die durch die Permutation (x,y) zu Verbindungen (b_0,b_3) werden. Das Verfahren ermittelt demnach entlang der Zeilen und Spalten x und y die Kostendifferenz der Indexpermutation.

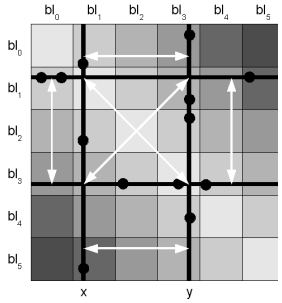

Abbildung 5: Kosten-gradient bei einer Abbildungspermutation

Weiterhin ist aufgrund der skalierbaren Laufzeit der Einsatz einer *Hill-Climbing* (dt.: Bergsteiger)-Meta-Heuristik möglich, welche den Algorithmus mehrfach mit verschiedenen Initial-Mappings startet, um die Abhängigkeit der Ergebnisse von der Qualität der initialen Daten zu reduzieren.[6]

Zur weiteren Optimierung wurde das Verfahren in seiner Suchbreite so erweitert, dass in jeder Iteration n verschiedene Indexpermutationen ermittelt und bewertet werden und die jeweils beste ausgewählt wird.

Außerdem wurde eine *Simulated-Annealing* (dt.: Simuliertes Erstarren) -Meta-Heuristik implementiert, welche mit einer sinkenden Wahrscheinlichkeit p auch verschlechternde Indexpermutationen erlaubt, um das Verlassen lokaler Optima zu ermöglichen.[6]

Der Algorithmus wurde aufgrund seiner geeigneten Struktur exemplarisch mittels *Multi-Threading* in einer Master-Slave-Architektur parallelisiert. Ein Anzahl von Threads ermitteln jeweils gleichzeitig eine neue zufällige Indexpermutation sowie deren Kostengradient. Ein Master-Thread organisiert die Verteilung der Aufgaben und ermittelt die beste verfügbare Permutation. Auf diese Weise ist der aufwändigste Abschnitt des Algorithmus parallelisiert.[7]

4) Verbessernde Rücksprung-Suche

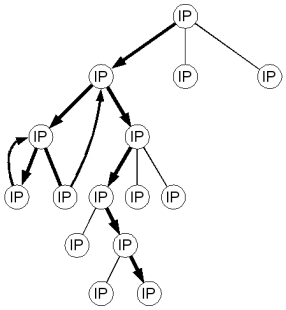

Abbildung 6: Rücksprünge im Suchbaum

Die verbessernde Rücksprung-Suche basiert ebenfalls auf dem Prinzip der Indexpermutation. Für eine gegebene Abbildung werden für einen Iterationsschritt die besten n Indexpermutationen ermittelt und die jeweils beste ausgewählt. Im Fall keiner weiteren Verbesserung springt der Algorithmus im Suchbaum zurück, um die Suche an anderer Stelle mit der nächstbesten Permutation fortzusetzen. Abbildung 6 zeigt schematisch den Pfad des Algorithmus in einem Suchbaum aus Indexpermutationen (*IP*) sowie dessen Rücksprünge im Fall

keiner weiteren Verbesserung. Das Verfahren ist als Greedy-Algorithmus einzustufen.[2]
Zur Optimierung wurden verschiedene Rücksprung-weitenregelungen eingeführt, um den Suchraum in kürzerer Zeit abzudecken:

- $t_{next} = t - a$ Rücksprung um *a* Rekursionstiefen
- $t_{next} = t / a$ skalierter Rücksprung
- $t_{next} = random(0 .. t-1)$ randomisierter Rücksprung

Aufgrund der geringen Änderungen der Abbildung innerhalb eines Iterationsschrittes wurde der Algorithmus um ein Verfahren erweitert, das nur die möglichen Indexpermutationen, die während der letzten Iteration beeinflusst worden, erneut evaluiert.
Um auftretende Zyklen zu vermeiden, wurde eine *Tabu*-Meta-Heuristik eingeführt, die bereits ausgeführte Indexpermutationen speichert und für die nächsten Iterationen verbietet.[6]

5) Genetischer Algorithmus

Mit einer Population aus Individuen, deren Genome die Neuronenabbildung darstellen und deren Fitness die Kosten der Abbildung repräsentieren, steht ein Set an Lösungen zur Verfügung, die durch genetische Operationen iterativ verbessert werden. Der prinzipielle Ablauf läßt sich wie folgt angeben:

```
P_i = InitialPopulation()
  P_i+1 = Duplikation(P_i)
  P_i+1' = Rekombination(P_i+1)
  P_i+1* = Mutation(P_i+1')
  P_i = Selektion(P_i+1*)
bis Abbruchkriterium erfüllt
```

Der Algorithmus terminiert nach einer gegeben Anzahl von Iterationen bzw. Generationen in denen keine Verbesserung erzielt wurde und ist als evolutionäres Verfahren einzustufen.[2][3][5][8]
Die Duplikation, bzw. Rekombination ist durch drei verschiedene Verfahren realisiert [4]:

- *Vermehrung durch Kopieren*: ein zufällig ausgewähltes Individuum wird in die Folgegeneration kopiert
- *Vermehrung durch stochastische Rekombination*: die Gene zweier zufällig ausgewählter Individuen werden zufällig miteinander zu einem neuen Individuum kombiniert. Invalide Gene, wie die Abbildung zweier Neuronen auf das gleiche Hardware-Element werden durch einen nachfolgenden Korrekturschritt behoben. Ein Beispiel ist in Abbildung 7 dargestellt.

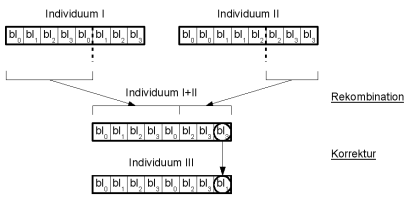

Abbildung 7: Stochastische Rekombination

- *Vermehrung durch selektive Rekombination*: die Gene zweier zufällig ausgewählter Individuen werden selektiv zu einem neuen Individuum kombiniert, indem das jeweils bessere Gen der beiden Eltern ausgewählt wird. Die Evaluierung der einzelnen Gene erfolgt gemäß ihrem Beitrag zur Fitness des Individuums. Demnach sind Neuronenabbildungen, die geringe Kosten verursachen die besseren Gene. Abbildung 8 zeigt schematisch die selektive Rekombination zweier Individuen.

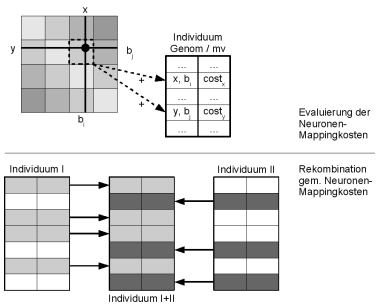

Abbildung 8: Selektive Rekombination

Die Mutation ist durch mehrere zufällige Indexpermutationen, wie bereits unter *c)* vorgestellt realisiert. [4]
Zur Reduktion der aktuellen Generation nach der Vermehrung und Mutation auf die Ursprungsgröße *g* werden zwei verschiedene Selektionsverfahren eingesetzt [4]:

- *Rang-Selektion*: Die Individuen werden gemäß ihrer Fitness geordnet. Anschließend werden die *g* besten Individuen ausgewählt, welche die Population der nächsten Generation darstellen.
- *Stochastische Selektion*: Die Individuen werden gemäß ihrer Fitness geordnet. Begonnen mit dem besten wird das aktuelle Individuum mit einer Wahrscheinlichkeit p_{sel} aus der Liste entfernt und in die nächste Generation übernommen, bzw. mit einer Wahrscheinlichkeit $1\text{-}p_{sel}$ übersprungen und zum nächstbesten Individuum übergegangen. Das Verfahren endet nach *g* Listendurchläufen.

Weiterhin wurde zur Erhaltung der genetischen Diversivität eine *Fitness-Sharing* (dt.: Fitness-Teilung) -Meta-Heuristik genutzt, welche die Fitness ähnlicher Individuen reduziert, um das Verlassen lokaler Optima zu ermöglichen. [5][8]

6) Segmentierungsverfahren

Das Segmentierungsverfahren zur Lösung des multikriteriellen Optimierungsproblems wurde auf der Überlegung aufgebaut, dass letztlich eine Zuweisung eines abstrakten Hardware-Elements zu je einer Neuronengruppe erfolgen soll, deren Neuronen möglichst viele Eigenschaften teilen. Als gemeinsame Eigenschaften können hierbei z.B.

- die synaptische Konnektivität
- gleiche Parameter
- gemeinsame sendende / empfangende Neuronen

angesehen werden. Ähnlich den Algorithmen zur automatischen Darstellung von Graphen werden die Neuronen als Punkte in einem n-dimensionalen Raum modelliert, nachfolgend *Neuronenpunkte* genannt, wobei die Entfernung zwischen zwei Punkten mit zunehmender Anzahl gemeinsamer

Eigenschaften abnehmen soll. Um die Selbstorganisation des Modells zu ermöglichen, werden verschiedene Kräfte eingeführt, die jeweils den Einfluss von Eigenschaften, die zwei Neuronen teilen oder nicht, repräsentieren. Der Betrag der Kräfte zwischen den Neuronenpunkten ist dabei von

- der Entfernung zueinander
- den Zuständen des Systems bzw. der Neuronenpunkte
- Parametern, welche die Wichtung der geteilten Eigenschaften hinsichtlich der Gesamtoptimierung angeben

abhängig. Schrittweise werden gem. diesen Kräften die Neuronenpunkte in dem n-dimensionalen Raum bewegt bzw. die Kräfte aktualisiert. Das Ergebnis ist eine Anordnung der Punkte so, dass Neuronen mit vielen gemeinsamen Eigenschaften nah beieinander bzw. mit wenigen geteilten Eigenschaften weiter voneinander entfernt positioniert sind. So ist eine Extraktion der Neuronengruppen möglich. Abbildung 9 zeigt an einem Beispiel die Segmentierung von sechs Neuronenpunkten gem. ihrer Konnektivität, sodass eine Aufteilung auf zwei Blöcke b_0 und b_1 möglich ist.

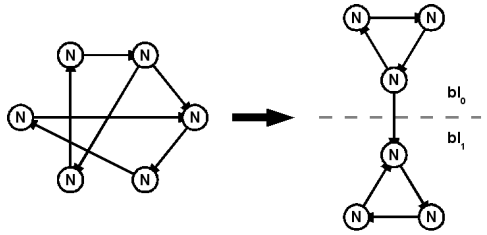

Abbildung 9: Netzsegmentierung

Eine mögliche Skalierung der anziehenden Kräfte f_{con} bei synaptischer Konnektivität zweier Neuronenpunkte und f_{nocon} bei keiner Konnektivität kann wie folgt angegeben werden:

$$f_{con} = par_{con} \cdot dist^2$$
$$f_{nocon} = par_{nocon} \cdot \frac{1}{dist^2} \quad (7)$$

par_{con} und par_{nocon} sind jeweils Parameter, die den Einfluss der Eigenschaft hinsichtlich der Segmentierung angeben. *dist* ist die Entfernung der Neuronenpunkte im n-dimensionalen Raum zueinander. Der Algorithmus realisiert weitere Kräfte, welche die Optimierungsziele des Verfahrens modellieren.

Neben der Gruppierung der Neuronenpunkte gem. ihren Eigenschaften muss parallel eine Zuordnung zu den Hardwareblöcken so erfolgen, dass deren Beschränkungen nicht verletzt werden. Zu diesem Zweck wurde für jeden Hardwareblock ein sog. *Clusterpunkt* eingeführt, was die Integration weiterer Kräfte zwischen Cluster- und Neuronenpunkten bedeutet. In jedem Iterationsschritt wird jeder Neuronenpunkt dem nächsten Clusterpunkt zugeordnet, was die Abbildung der Neuronen auf die Hardware-Blöcke, und damit das Mapping in dieser Hardware-Ebene darstellt. Gleichzeitig wird anhand dieser Zuordnung, das sog. Gewicht jedes Clusterpunktes bestimmt. Das Gewicht beeinflusst maßgeblich die Skalierung der vom Clusterpunkt verursachten Kräfte und sorgt für eine gleichmäßige Aufteilung der Neuronenpunkte auf die Hardware-Blöcke.

Um das System nicht expandieren, kollabieren, stagnieren oder oszillieren zu lassen, ist eine gezielte Parametrisierung aller Kräfte nötig. Zu diesem Zweck wurden u.a. sog. Cluster-Halos eingeführt, welche die für die Segmentierung irrelevanten starken Kräfte isoliert, gleichzeitig aber die Plastizität des Systems sichert.

Der Algorithmus wurde aufgrund seiner geeigneten Struktur exemplarisch mittels Multi-Processing in einer Master-Slave-Architektur parallelisiert. Die physikalische Trennung der Prozesse durch separate Speicher und Prozessoren macht eine spezielle Kommunikationsstruktur erforderlich, die mit einer LAM (*Local Area Multicomputing*) / MPI (*Message Passing Interface*)-Implementierung umgesetzt wurde.[7]

Nach der Initialisierung der Arbeitsprozesse mit nötigen Daten, wie z.B. der Verbindungsmatrix, verteilt der Hauptprozess in jeder Iteration die Aufgaben zur gleichzeitigen Evaluierung der Kräfte der Neuronen- und Clusterpunkte. Diese Kräfte werden vom Hauptprozess gesammelt und skaliert und die Parameter für die nächste Iteration an die Arbeitsprozesse übermittelt.

7) Diagonalisierungsverfahren nach CutHill McKee

Bei geeignter Indizierung der Hardwareblöcke ist die Diagonalisierung der Verbindungsmatrix sinnvoll. Zu diesem Zweck wurde das Verfahren nach CutHill-McKee eingesetzt.[9]

C. Synapsenverteilungsverfahren

Das Synapsenverteilungsverfahren bildet nach abgeschlossener Zuordnung aller Neuronen zu Hardware-Elementen die Synapsen auf die niedrigst möglichen Kommunikations-elemente ab. In Abhängigkeit der topologischen Entfernung der beiden beteiligten Neuronen sowie den Beschränkungen der Kommunikation, z.B. durch maximale Ein- und Ausgänge werden die Synapsen im hierarchischen Hardware-Modell (siehe Abbildung *3*) von unten nach oben eingeordnet.

V. Auswertung der Algorithmen

A. Test-Szenarien

Zur Analyse der Optimierungsalgorithmen wurden verschiedenen Test-Szenarien entworfen, die jeweils aus einem biologischen und einem Hardware-Modell bestehen, die aufeinander abgebildet werden sollen. Abbildung 10 zeigt die drei Verbindungsmatrizen der verwendeten biologischen Modelle, Abbildung 11 die schematische Darstellung einiger Hardware-Modelle.

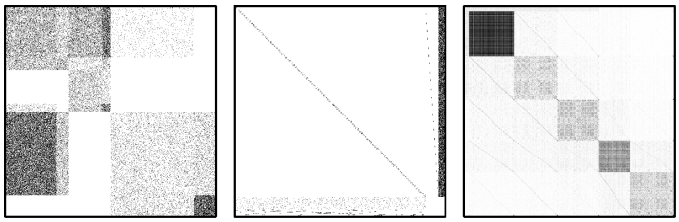

Abbildung 10: biologische Modelle

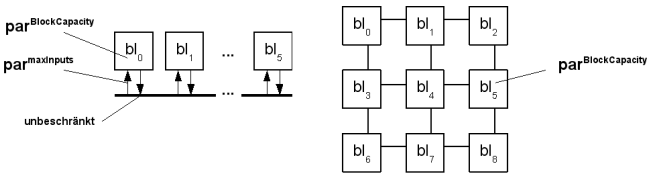

Abbildung 11: Hardware-Modelle

B.Ergebnisse

Zum Vergleich der Algorithmen wurden u.a. die Laufzeit sowie die relative Kostenoptimierung der Algorithmen in jedem Test-Szenario erhoben. Weiterhin wurde die Untersuchung für viel versprechende Algorithmen vertieft und spezielle Daten zur Verfahrensoptimierung ermittelt.
Die erzielten relativen Kosten-Optimierungen sind für alle Verfahren von den Modellgrößen unabhängig. Zusammenfassend lässt sich für die Algorithmen sagen:

1) Vollsuche

Das Verfahren konvergiert für sehr kleine Modelle aufgrund der exponentiell zunehmenden Möglichkeiten nicht in annehmbarer Zeit und wurde daher von der weiteren Untersuchung ausgeschlossen.

2) Einsortierungsverfahren

Die Laufzeit nimmt mit wachsenden Modellgrößen vergleichsweise sehr stark zu. Die erzielten Kostenoptimierungen sind aufgrund der zeitig und damit suboptimal abgebildeten Neuronen als vergleichsweise gering zu bewerten. Das Verfahren ist daher ungeeignet.

3) Zufällig permutierende Suche und Genetische Suche

Im Fall effizient ermittelbarer Kostengradienten nimmt die Laufzeit mit wachsenden Modellgrößen vergleichsweise durchschnittlich zu. Bei komplexen Hardware-Modellen hingegen steigt die Laufzeit vergleichsweise sehr stark an. Die erzielten Kostenoptimierungen sind vergleichsweise gut bis sehr gut. Die Verfahren sind daher für große Netze, aber nur bestimmte Hardware-Modelle geeignet.
Die Fitness-Sharing- und Rekombination-Operatoren der Genetischen Suche verschlechtern die Konvergenz, bzw. die Leistung des Algorithmus erheblich. Die Wahl der beiden Selektionsverfahren ist für die Leistung irrelevant.

4) Verbessernde Rücksprung-Suche

Die Laufzeit nimmt mit wachsenden Modellgrößen vergleichsweise sehr stark zu. Die erzielten Kostenoptimierungen sind als vergleichsweise sehr gut zu bewerten. Das Verfahren ist daher für die Optimierung kleiner Modelle geeignet.

5) Segmentierungsverfahren

Die Laufzeit nimmt mit wachsenden Modellgrößen vergleichsweise langsam zu. Die erzielten Kostenoptimierungen sind als vergleichsweise gut bis sehr gut zu bewerten. Das Verfahren ist daher für die Optimierung großer bis sehr großer Modelle geeignet.

6) Diagonalisierungsverfahren CutHill-McKee

Die Laufzeit nimmt mit wachsenden Modellgrößen vergleichsweise sehr langsam zu. Die erzielten Kostenoptimierungen sind in Abhängigkeit vom verwendeten Hardware-Modell als durchschnittlich bis sehr gering zu bewerten. Das Verfahren ist daher für die Optimierung großer bis sehr großer Netze, aber nur sehr spezieller Hardware-Modelle geeignet.

VI. Zusammenfassung

Zur Erstellung und Analyse der multikriteriellen Optimierungsalgorithmen wurde zunächst eine formale Problembeschreibung sowie geeignete Kostenfunktionen und Modellbeschreibungen als Graphen gefunden.
Darauf aufbauend wurden in einem Framework der gesamte Mapping-Prozess umgesetzt sowie verschiedene Algorithmen entwickelt und optimiert.
In verschiedenen Test-Szenarien wurden die Algorithmen anschließend systematisch hinsichtlich Kriterien wie Laufzeit und relative Kostenoptimierung untersucht.
Dabei kristallisierten sich die zufällig permutierende Suche, die genetische Suche sowie das speziell entwickelte Segmentierungsverfahren als leistungsstärkste Algorithmen heraus.
Für die vollständige Abbildung der Netze auf die VLSI-Hardware sind die gefunden Algorithmen zu verbessern und zu erweitern sowie die erstellten Modellbeschreibungen auszubauen. Weiterhin müssen neue Algorithmen erstellt und evaluiert werden um die Abbildungsbeschreibung zu verfeinern und zu vervollständigen.

VII. Literatur

[1] C. Mayr, M. Ehrlich, S. Henker, “Mapping Complex, Large - Scale Spikung Networks on Neural VLSI”. Department of Electrical Engineering, University of Technology Dresden, Germany, 2007
[2] M. Ehrgott, X. Gandibleux, “A survey and annotated bibliography of multiobjective combinatorial optimization”. Department of Engineering Science, University of Auckland, New Zealand, 2007
[3] D. Whitley, “A genetic algorithm tutorial. Computer”. Science Department, Colorado State University, USA, 1994
[4] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, S. Dizdarevic, “Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators”, Depr. of Coumpter Science and Articial Intelligence, University of the Basque Country, Spain.
[5] A. Jaszkiewicz, “Genetic local search for multi-objective combinatorial optimization”. Institute of Computer Science, Pozna´n University of Technologie, Poland, 2000
[6] D.F. Jones, S.K. Mirrazavi, M. Tamiz, “Multi-objective meta-heuristics: An overview of the current state-of-the-art”. School of Computer Science and Mathematics, University of Portmouth, UK, 2001
[7] T. Letschert, “Nebenläufige und Verteilte Programmierung”. Logos Verlag Berlin, 1998
[8] D.A. Van Veldhuizen, G.B. Lamont, “Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art”. Air Force Resarch Laboratory, Optical Radiation Branch, Brooks AFB, TX 78235, USA, 2000
[9] Y. Saad, “Iterative Methods for Sparse Linear Systems”. PWS Boston, 1996