Multikriterielle Evolution adaptiver eingebetteter Systeme*

Paul Kaufmann

Institut für Informatik
Universität Paderborn
paul.kaufmann@uni-paderborn.de

Abstract: Die Kombination rekonfigurierbarer elektronischer Bausteine mit den Methoden der künstlichen Intelligenz erschafft für eingebettete Systeme einen eleganten und uniformen Ansatz zur Adaptation an Veränderungen der Umwelt, Defekte der Hardware und Variationen in den Anforderungen an Systemressourcen. Dieses, unter dem Begriff Evolvable Hardware bekannte Prinzip, hat auf eindrucksvolle Weise das Entdecken neuer Entwurfsprinzipien und neuartiger sowie leistungsfähiger Lösungen für bestehende Ingenieursaufgaben aufgezeigt.

In dieser Arbeit präsentieren wir einen ganzheitlichen Ansatz für Evolvable Hardware und stellen unsere Ergebnisse auf dem Gebiet des effizienten Entwurfs Boolescher Schaltungen vor. Die entwickelten Methoden werden für die Evolution von hardwarebasierten Mustererkennungsarchitekturen sowie Prozessoroptimierung eingesetzt.

1 Das Evolvable Hardware Prinzip

Evolvable Hardware ist die immerwährende Optimierung eines rekonfigurierbaren Systems. Vorgestellt 1993 von Higuchi, de Garis u.a. in [HNT+93], liegt Evolvable Hardware folgende Funktionsweise zu Grunde:

1. Basierend auf dem iterativen Zyklus eines evolutionären Algorithmus, startet die lebenslange Optimierung mit einem oder mehreren Bauplänen des zu adaptierenden Systems. In der Terminologie der evolutionären Algorithmen wird ein Bauplan als Genotyp oder Individuum und die Gesamtheit der Individuen als Population bezeichnet. Das zu optimierende System ist bereits mit einer vorher evolvierten oder entwickelten Lösung im Einsatz. Der geschilderte Ablauf wird in der Abbildung 1(a) anhand der Adaptation eines Bildkomprimierers veranschaulicht.

2. In der nächsten Phase werden die Individuen bezüglich ihrer Leistungsfähigkeit (Fitness) bewertet. Dazu bedarf es aktueller Messwerte, wie etwa der neusten Kameraaufnahmen, wie in Abbildung 1(b) dargestellt. Mit diesen Eingabedaten kann für jeden Genotyp die Kompressionsrate berechnet werden, die z.B. vom aktuellen Ausleuchtungsgrad abhängt (Abbildung 1(c)).

*Englischer Titel der Dissertation: “Adapting Hardware Systems by Means of Multi-Objective Evolution” [Kau13]

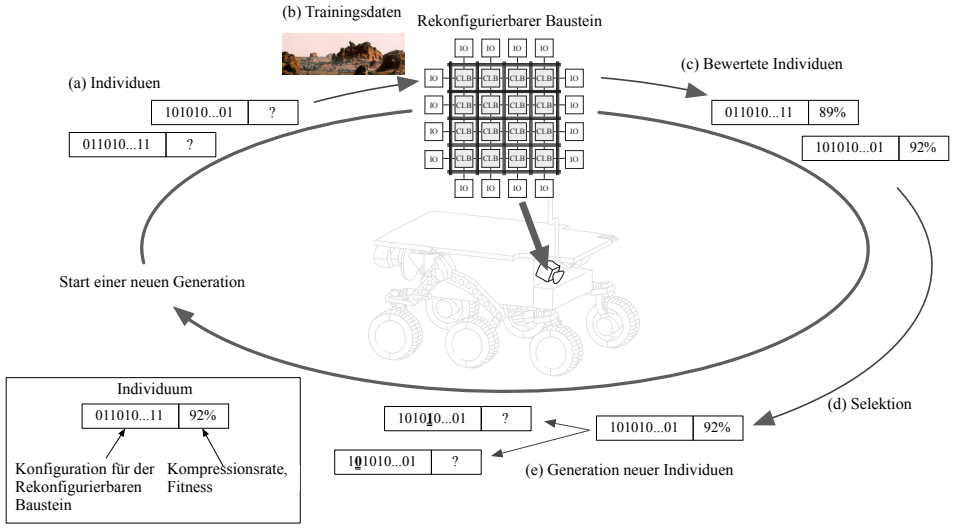

Abbildung 1: Adaptiver Bildkompressor mithilfe von Evolvable Hardware.

Bei der Auswertung der Fitness gibt es einen entscheidenden Punkt: Wird die Fitness auf der selben Hardware ausgewertet, die später auch das Individuum operativ ausführt, ist auch eine Kompensation von Hardwaredefekten möglich. Dies liegt daran, dass ein evolutionärer Algorithmus in seiner klassischen Ausprägung stets die Funktionalität einer Lösung zum Ziel hat. Die innere Struktur der Lösung unterliegt dabei dem Optimierungsprozess. Veränderungen und Fehler in den Rechenressourcen können somit implizit berücksichtigt und kompensiert werden. Die Auswertung der Fitness auf der Zielplattform erlaubt auch die Evolution von Lösungen für Rechenplattformen, die nur angenähert formalisiert werden können und die Schwankungen im Herstellungsprozess unterliegen. Solche Plattformen sind z.B. rekonfigurierbare Transistor- und Operationsverstärkerbausteine.

3. Nun kommen die Prinzipien der biologischen Evolution zum Zuge. Geleitet von der Fitness, werden zielgerichtet bestehende und gute Eigenschaften verschiedener Individuen kombiniert (das Prinzip der geschlechtlichen Fortpflanzung, Rekombination) sowie bestehende Lösungen durch Hinzufügen fundamental neuer Eigenschaften mittels zufälliger Veränderungen des Genotyps (Mutation) sukzessiv verbessert (Abbildung 1(d),(e)). Mit der Erzeugung einer neuen Population schließt sich der Optimierungszyklus eines evolutionären Algorithmus und im nächsten Schritt kann die Fitness der neuen Individuen bewertet werden.

Das Evolvable Hardware Prinzip eignet sich insbesondere für die Adaptation von Zielfunktionen, die eine graduelle Funktionsgüte haben. Das können Funktionen sein, deren interne Struktur von der Verteilung der Eingabedaten abhängt und Funktionen, die zwar eine perfekte Lösung besitzen, aufgrund der endlichen Ressourcen aber mit einer kleinen und suboptimalen Realisierung auskommen müssen. Beispiele dafür sind Mustererkennungsalgorithmen, Regel- und Einbahnfunktionen.

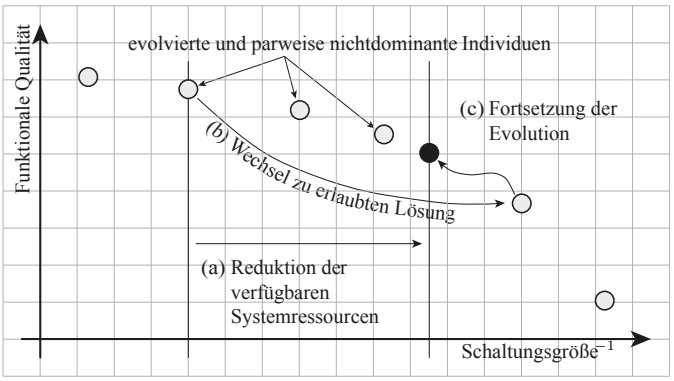

Abbildung 2: Kompensation rascher Änderungen mithilfe multikriterieller Algorithmen.

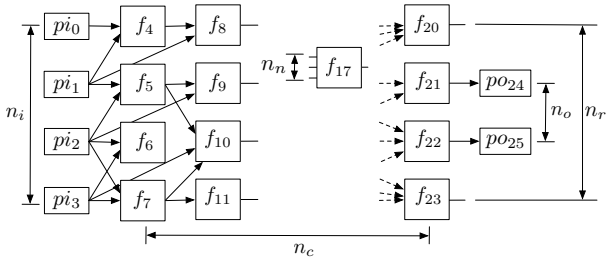

Abbildung 3: Cartesisches Genetische Programmieren.

Unsere Motivation für eine holistische Realisierung von Evolvable Hardware gründet in der Einsicht, dass es mehrere Zeitebenen geben kann, in denen sich ein autonomes System anpassen können muss. Während kontinuierliche Veränderungen mit niedrigen Veränderungsraten, üblicherweise Veränderungen in der Systemumwelt, durch kontinuierliche Optimierung kompensiert werden können, bedürfen rasche Veränderungen, oft Veränderungen in den Systemressourcen, präevolvierter Lösungen. Um einen autonomen Betrieb des adaptiven Systems zu gewährleisten, können in diesem Fall Paretobasierte multikriterielle evolutionäre Algorithmen eingesetzt werden. Diese können zur Laufzeit Lösungen evolvieren, die verschiedenartig bezüglich der sich rasch verändernden Zielfunktionen sind. Ein Beispiel ist in der Abbildung 2 dargestellt. Kreise symbolisieren Lösungen, die sich bezüglich der funktionalen Qualität und Größe unterschieden. Bei einer plötzlichen Reduktion der verfügbaren Systemressourcen (Abbildung 2(a)) wird die gerade operative Lösung durch eine Lösung ersetzt, die die neue Vorgaben einhält (Abbildung 2(b)). Danach wird der Optimierungsprozess fortgesetzt und es wird versucht, die funktionale Qualität unter Ausnutzung der noch verfügbaren Ressourcen zu maximieren (Abbildung 2(c)).

Der Fokus unserer Arbeit liegt auf digitaler Evolvable Hardware unter Verwendung von feldprogrammierbaren Logikbausteinen (FPGA). Dazu bedarf es zunächst einer Formalisierung des FPGA Berechnungsmodells. Wir verwenden hier die kartesische genetische Programmierung (CGP) nach Miller und Thomson [MT00], vorgestellt in Abbildung 3. Es besteht, ähnlich einem FPGA, aus gitterartig angeordneten Funktionsblöcken. Um die

Komplexität der Funktionsauswertung vertretbar zu halten, bildet CGP nur den kombinatorischen Fall ab. Dazu dürfen Funktionsblöcke keine Speicher und Signalpfade keine Zyklen enthalten. Leitungen dürfen nur Funktionselemente einer Spalte mit Funktionselementen in den darauf folgenden Spalten verbinden. Die maximale Leitungslänge l kann beschränkt werden.

In FPGA basierten Systemen können rasche Veränderungen durch Veränderungen in der Auslastung des FPGAs oder durch Neuzuteilung der FPGA Ressourcen für die aktiven Applikationen verursacht werden. Damit sind Schaltungsgröße und Verarbeitungsgeschwindigkeit Optimierungsziele, bezüglich derer verschiedenartige CGP Schaltung evolviert werden müssen.

In dieser Arbeit haben wir uns folgenden Herausforderungen gestellt: Zunächst benötigt das EHW Konzept einen effizienten, Paretobasierten multikriteriellen evolutionären Algorithmus (MOEA). Ein solcher Algorithmus gehört üblicherweise zu der Familie der globalen Optimierer. Dazu setzen diese Algorithmen den Rekombinationsoperator ein, um entfernte Bereiche des Suchraums erkunden und Informationsdrift zwischen den Individuen realisieren zu können. Der Suchraum der CGP Schaltungen ist aber hochgradig nichtlinear und begünstigt lokale Sucher, die sich ausschließlich auf den Mutationsoperator verlassen. Für den Einsatz Paretobasierter MOEAs für die Evolution von CGP Schaltungen ist somit ein effektiver Rekombinationsoperator notwendig. Unsere Arbeiten auf diesem Gebiet sowie zur Verbesserung der Skalierbarkeit der Evolution von CGP Schaltungen werden im Kapitel 2 zusammengefasst.

Des Weiteren lässt sich die Effizienz evolutionärer Methoden anhand synthetischer Testfunktionen abschätzen, die tatsächliche Verwendbarkeit und Nützlichkeit einer randomisierten Heuristik ist aber nur an realen Anwendungen demonstrierbar. Aus diesem Grund evaluieren wir im Kapitel 3 unsere Methoden an den Beispielen eines Mustererkenners für Prothesensteuerungen und an der Optimierung eines Prozessorcaches.

2 Effiziente Multikriterielle Evolutionäre Algorithmen für CGP

In diesem Kapitel fassen wir unsere Arbeiten zum CGP Rekombinationsoperator, der automatischen Akquise und Wiederverwendung von CGP Subfunktionen, der Priorisierung von Optimierungszielen und Periodisierung von Suchalgorithmen zusammen.

2.1 Strukturbasierter CGP Rekombinationsoperator

Im CGP Modell kann eine Funktion auf verschiedene Arten kodiert werden. Z.B. lässt sich die Funktion $\mathrm{po}_{24} = (\mathrm{pi}_0 \text{ AND } \mathrm{pi}_1)$ bei geeignet gewähltem Parameter l durch jeden der Funktionsblöcke im CGP Gitter in der Abbildung 3 berechnet werden. Die räumliche Anordnung der Funktionsblöcke, obwohl irrelevant für die berechnete Funktion, wird mitkodiert und mitevolviert und macht bei strukturellen Manipulationen einer Lösung, z.B. bei Austausch von Teillösungen, eine Revision der Abbildung der Funktion auf das CGP

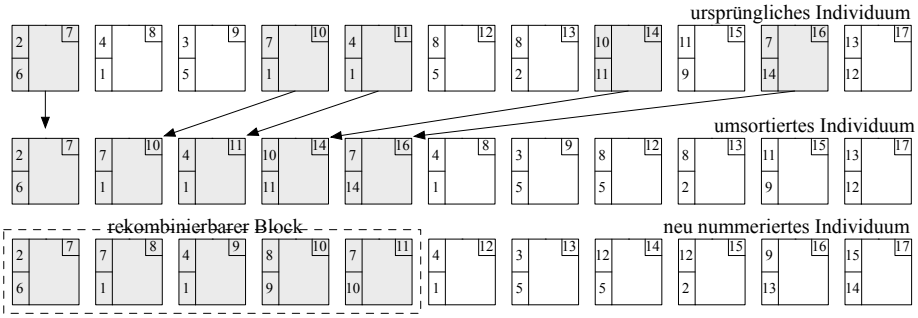

Abbildung 4: CGP Rekombination. Die Nummer in der rechten oberen Ecke eines Funktionsblocks ist seine laufende Nummer. Die Nummern auf der linken Seite sind Nummern funktional abhängiger Blöcke (z.B. grau schattiert).

Gitter notwendig. Da dieser Prozess nicht trivial ist, wurde er bisher nicht untersucht.

Möchte man nun einen Paretobasierten multikriteriellen Evolutionären Algorithmus auf CGP anwenden, ist die Implementierung eines Rekombinationsoperators, der für jede rekombinierte Funktion eine neue Abbildung auf das CGP Gitter berechnet, unumgänglich. Um dies zu bewerkstelligen, betrachten wir nur den eindimensionalen CGP Raum ($n_c = 1, l = \infty$). Für die Rekombination wird für ein Elterindividuum ein Funktionsblock (Wurzel) und die Anzahl der auszutauschenden Funktionsblöcke n zufällig gewählt. Eine Breitensuche markiert maximal n funktional abhängige Vorgängerknoten der Wurzel und das CGP Individuum wird so permutiert, dass die markierten Funktionsblöcke einen konsolidierten Block bilden (Abbildung 4). Das ist immer möglich und das resultierende Individuum ist stets eine gültige CGP Schaltung. Rekombination ist somit auf den Austausch solcher Blöcke zurückführbar.

Tabelle 1: Berechnungsaufwand für strukturbasierten Rekombinationsoperator sowie struktur- und altersbasierte Akquisition von Subfunktionen. Hervorgehobene Zahlen bedeuten eine Verbesserung. $1 + 4$ ES ist der Refernzalgorithmus.

	$1+4$ ES	Rekombination		Altersbasiert		Strukturbasiert	
	absolut	absolut	relativ	absolut	relativ	absolut	relativ
2×2 mul	66.623	64.111	$\mathbf{-3.8\%}$	51.961	$\mathbf{-22.0\%}$	49.052	$\mathbf{-26.4\%}$
3×3 mul	8.840.574	2.518.964	$\mathbf{-71.5\%}$	6.001.917	$\mathbf{-32.1\%}$	3.638.120	$\mathbf{-58.9\%}$
85% EMG	18.260	19.859	$+8.8\%$	14.743	$\mathbf{-19.3\%}$	23.855	$+30.7\%$
95% EMG	510.147	576.988	$+13.1\%$	314.311	$\mathbf{-38.4\%}$	873.319	$+71.2\%$

Die Auswertung des Rekombinationsoperators ist in der Tabelle 1 vorgestellt. Als Vergleichsmetrik dient der Berechnungsaufwand nach Koza [Koz94] und als Testfunktionen verwenden wir 2×2 und 3×3 Multiplizierer sowie Mustererkennungsfunktionen (EMG), deren Evolution beim Erreichen von 85% bzw. 95% der Trainingsgüte gestoppt und als Erfolg gewertet wird. Aus der Tabelle lässt sich schlussfolgern, dass für Multiplizierer, die viele repetitive Substrukturen besitzen, der strukturbasierte Rekombination sehr effizient ist wogegen für die Mustererkennungsfunktionen, die weniger ausgeprägte innere Struktur

aufweisen, der Operator im Vergleich zu lokaler Suche im Nachteil ist. Eine detailliertere Auswertung ist in [Kau13] zu finden.

2.2 Automatische Identifikation und Wiederverwendung von Subfunktionen

Die zuvor implementiere Funktion zur Extraktion funktional abhängiger Substrukturen kann dazu verwendet werden, die Skalierbarkeit evolutionärer Algorithmen zu verbessern. Die Idee ist es, häufig vorkommende Substrukturen automatisch zu identifizieren und als primitive Funktionale, Module, der Menge der erlaubten CGP Blockfunktionen hinzuzufügen und wiederzuverwenden [Koz94]. Dazu haben wir unseren strukturbasierten Rekombinationsoperators erweitert und anhand gleicher Testfunktionen wie im vorigen Experiment untersucht. Die ursprünglichen Ergebnisse werden auch in diesem Experiment bestätigt (Tabelle 1). Die Evolution von arithmetischen Schaltungen kann deutlich verbessert werden wogegen für Mustererkennugsfunktionen der strukturbasierte Operator wenig effektiv bleibt.

Um auch für nichtarithmetische Funktionen über einen gut skalierbaren evolutionären Operator zu verfügen, entwickelten wir ein altersbasierters Verfahren zur automatischen Identifikation von Modulen. Dieses Verfahren geht davon aus, dass, ähnlich der Entstehung von Organen in der biologischen Evolution, über viele Generationen unverändert gebliebene Substrukturen implizit zum Erfolg eines Individuums beitragen. Für die Umsetzung des Verfahrens bekommt jeder CGP Funktionsblock einen Zähler, der bei der Erzeugung einer neuen Generation inkrementiert wird. Wird ein Funktionsblock dagegen von einem Mutationsoperator verändert, wird sein Zähler auf Null gesetzt. Für strukturbehaftete Zielfunktionen, wie etwa Multiplizierer, ist die altersbasierte Modulerzeugung zwar besser als der Referenzalgorithmus, reicht aber an die Leistung der strukturbasierten Modulerstellung nicht heran (Tabelle 1). Dagegen ist die altersbasierte Modulerstellung deutlich besser als die strukturbasierte Modulerstellung und besser als der Referenzalgorithmus für die Evolution von Mustererkennungsfunktionen mit wenig innerer Struktur.

2.3 Priorisierung von Optimierungszielen

Im Suchraum der CGP Schaltungen ist die Skalierbarkeit Paretobasierter MOEAs oft dramatisch schlechter als die Skalierbarkeit lokaler Verfahren, wie z.B. der (1+4) Evolutionärer Strategien (ES). Um die Skalierbarkeit Paretobasierter MOEAs zu verbessern, haben wir zunächst versucht, die impliziten Abhängigkeiten der Optimierungsziele Schaltungsgröße, Verarbeitungsgeschwindigkeit und funktionale Qualität auszunutzen, ohne dabei die Diversität der evolvierten Lösungsmengen zu beeinträchtigen. Dazu haben wir den populären Algorithmus SPEA2 [Kau13] erweitert, indem wir den Selektionsdruck für die Evolution neuer Individuen abhängig von einer Linearkombination aller Optimierungsziele gemacht und die Gewichte der Linearkombination in vorhergehenden Experimenten bestimmt haben. Dabei blieb der Diversitätsmechanismus von SPEA2 unangetastet.

Mit diesem Ansatz lässt sich die multikriterielle Evolution von CGP Schaltungen bereits entscheidend verbessern. Der neue Algorithmus ist nicht nur in der Lage, häufiger Individuen mit hoher funktionaler Qualität zu finden, sondern auch Lösungsmengen zu evolvieren, die in ihrer Diversität den Lösungsmengen der Referenzalgorithmen in Nichts nachstehen und sogar oft besser sind [Kau13].

2.4 Periodisierung Lokaler und Globaler Suche

Die Priorisierung von Optimierungszielen definiert eine Suchrichtung. Für einen universelleren Optimierer, der ohne eine vorgegebene Suchrichtung verschiedenartige nichtdominante Lösungsmengen zuverlässig finden kann, greifen wir zu der Idee der Algorithmusperiodisierung. Dazu definieren wir eine Sequenz aus Algorithmen und eine Wiederholungsfunktion f. Die Algorithmussequenz wird zyklisch ausgeführt wobei die Ausgabepopulation eines Algorithmus in der Sequenz die Eingabepopulation des nächsten Algorithmus wird. Abhängig von f und der Historie des Optimierungslaufs, wird ein Algorithmus einmal, mehrmals oder auch gar nicht ausgeführt.

Bei der Periodisierung von Paretobasierten MOEAs ist zu beachten, dass die erzeugten Lösungsmengen eine innere Struktur haben. Nicht-Pareto basierte Algorithmen bedürfen deswegen einer Anpassung. Da sich (1+4) ES als sehr effizient für CGP gezeigt haben, entschieden wir uns dieses lokale Suche Verfahren für die Periodisierung anzupassen. Im Folgenden unter dem Namen hybride ES (hES) geführte Algorithmus führt für jedes Individuum der Eingabepopulation einen (1+4) ES Schritt aus und selektiert aus den vier neuen Individuen und dem Elterindividuum exakt ein Individuum aus, das dominant unter diesen fünf Individuen ist oder bezüglich einer Ähnlichkeitsmetrik über alle Individuen als möglichst verschiedenartig eingestuft wird. Um neutrale Suche zu ermöglichen, wird das Elterindividuum nur dann in die neue Generation kopiert, wenn es strikt besser als seine Kinderindividuen ist.

In Experimenten zeigte sich ein ähnliches Bild wie bei der Priorisierung von Optimierungszielen. Die Periodisierung von lokalen und globalen Suchern ist vorteilhaft bezüglich der Evolution von funktional qualitativen Schaltungen ohne dabei negativ für die Verschiedenartigkeit der evolvierten Lösungsmenge zu sein.

3 Evolvable Hardware Anwendungen

Die vorgestellten Methoden zur effizienten Evolution von CGP Schaltungen haben wir auf Mustererkennungsalgorithmen für Prothesensteuerungen und Prozessorcaches angewendet. Beide Applikationen können von Laufzeitadaption profitieren, um z.B. die Klassifikationsgüte aufrechtzuerhalten oder die Ausführungszeit eines Prozessors für neue Applikationen und Datensätze zu minimieren.

3.1 Evolvierbare Klassifizierer

In Abbildung 5 ist ein CGP basierter Ensembleklassifizierer vorgestellt. Jedes Klassifikationsmodul k besteht aus mehreren CGP Schaltungen, die darauf trainiert sind, für Eingabevektoren aus der k-ten Kategorie eine "1" und sonst eine "0" zu berechnen. Das Klassifikationsmodul mit der größten Anzahl aktivierter CGP Schaltungen bestimmt das Ergebnis des Klassifizierers.

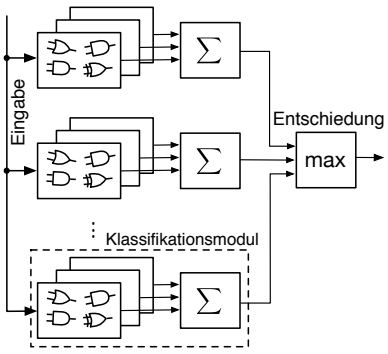

Abbildung 5: CGP basierter Ensembleklassifizierer.

Für einen realistischen Datensatz haben wir über 21 Tage in 121 Sitzungen Muskelspannungen des Unterarmes für elf Bewegungen aufgenommen. Dabei haben wir für die Merkmalsextraktion das gleitende Mittel verwendet, da dieses sich sehr effizient auf einem eingebetteten System implementieren lässt.

In initialen Experimenten konnten wir beobachten, dass die Klassifikationsgüte eines Mustererkennungsalgorithmus nachlässt, wenn dieser nur mit den Daten der ersten Sitzungen trainiert wird. Das macht ein periodisches Training mit aktuellen Daten notwendig. Ausgehend von dieser Beobachtung haben wir ein Evaluationsschema festgelegt, bei dem ein Algorithmus mit den Daten der Sitzungen $(i-5, \ldots, i-1)$ trainiert und mit den Datensatz i ausgewertet wird.

Die Tabelle 2 vergleicht unseren Ansatz mit den Methoden der k-nächsten-Nachbarn, Entscheidungsbäumen, neuronalen Netzen und Support Vektor Maschinen. Der Evolvable Hardware Klassifizierer reicht zwar nicht an die Leistung bester konventioneller Algorithmen heran, ist aber besser als die Entscheidungsbäume. Weiterhin können bei einer tatsächlichen Implementierung Eigenschaften wie effizientes Lernen, Kompaktheit und Energieeffizienz vor der absoluten (aber mindestens ausreichenden) Klassifikationsleitung ausschlaggebend sein.

Tabelle 2: Gemittelter Generalisierungsfehler.

	Fehlerrate
kNN	10.45
DT	17.91
MLP	10.44
SVM	**9.00**
CGP	16.48

In einem weiteren Experiment wurde die Laufzeitadaptionsfähigkeit des EHW Klassifizierers untersucht, indem man die Anzahl der Klassifikationsschaltungen in einem Klassifikationsmodul zur Laufzeit variierte und den Klassifizierer neu trainierte. Man konnte dabei die Beobachtung machen, dass der EHW Ensembleklassifizierer bereits mit sehr wenigen Klassifikationsschaltungen pro Klassifikationsmodul sehr gut diskriminieren kann und dass mehr Ressourcen zu einer schnelleren Wiederherstellung der Erkennungsraten führen können.

3.2 Evolvierbare Prozessorcaches

Eine Anwendung, die zunächst nicht strikt der Definition eines Evolvable Hardware System entspricht, ist das Optimieren einer Cachefunktion, die Speicheradressen eines Prozessors auf Cacheadressen abbildet. Führt man die Optimierung zur Laufzeit oder gar verteilt durch, entsteht ein Evolvable Hardware System das sich nicht wie bisher in der physikalischen Welt bewegt, sondern im globalen Computernetzwerk sein Habitat hat.

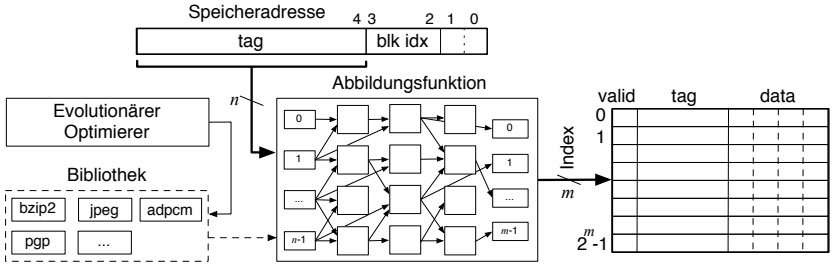

Abbildung 6: Das Prinzip eines evolvierbaren Prozessorcaches.

Die Idee evolvierbarer Cachefunktionen ist in der Abbildung 6 vorgestellt. Anstatt die Indexbits einer Speicheradresse für die Indizierung innerhalb eines Caches zu verwenden, werden die Tag- und Indexbits Eingaben einer rekonfigurierbaren CGP Schaltung, die mithilfe eines evolutionären Algorithmus optimiert wird. Die Schaltungsausgaben werden zur Cacheindizierung benutzt.

Tabelle 3: Verbesserung gegenüber einem konventionellen Cache in %.

	bzip2		jpeg	
	L1	L2	L1	L2
Ausführungszeit	5,84	7,95	14,31	12,96
Missrate	6,83	9,77	41,25	40,35
Energieverbrauch	5,71	7,83	16,43	14,46

Wir haben die Idee der evolvierbaren Caches an den Beispielen des bzip2 Komprimierers und des jpeg Kodierers evaluiert. Dazu haben wir in einer zyklengenauen Simulation für beide Anwendungen und vorher definierte Eingabedaten für entweder L1I und L1D oder L1I, L1D und L2 die Cachefunktionen evolviert und anschließend unter Benutzung von Testeingabedaten evaluiert. Die Speicherhierarchie des simulierten Systems bestand dabei aus einem Prozessor, einem L1 Instruktions- und Datencache (L1I, L1D), einem L2 Cache und dem Hauptspeicher. Die Ergebnisse sind in der Tabelle 3 zusammengefasst. Das wichtigste Ergebniss ist, dass die evolvierten Cachefunktionen sehr gut für unbekannte Eingabedaten generalisieren. Für bzip2 kann die Ausführungszeit und der Energieverbrauch um nahezu 8% gegenüber einem konventionellen System reduziert werden. Im jpeg Fall sind es sogar mehr als 14% und 16%. Die Trefferrate eines Caches kann bei bzip2 um 9,7% und bei um bis zu 41% verbessern werden. Die evolvierten Cachefunktionen sind im Durschnitt 13 bis 16 CGP Funktionsblöcke groß wobei der längste Pfad bis zu vier Funktionsblöcke enthalten kann.

4 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Entwicklung und Erprobung eines ganzheitlichen Konzepts für Evolvable Hardware. Die wesentlichen Ergebnisse hierbei sind effiziente multikriterielle Methoden für den automatischen Entwurf und die Optimierung Boolescher CGP Schaltungen sowie Anwendungen, die mithilfe dieser Methoden umgesetzt werden konnten und somit die Nützlichkeit des Evolvable Hardware Ansatzes verdeutlichen.

Im Kontext dieser Arbeit können folgende Schlussfolgerungen gemacht werden: Das Prinzip der nichtdeterministischen und kontinuierlichen Adaptation eines Systems beschränkt sich nicht nur auf autonome und adaptive eingebettete Systeme sondern kann für artfremde Anwendungen überraschend neuartige und innovative Lösungsansätze schaffen. Weiterhin kann das Evolvable Hardware Konzept im konventionellen Ingenieursbereich, trotz berechtigter Skepsis gegenüber nichtdeterministischer Adaptation, Lösungen schaffen und Anwendungen ermöglichen, die von traditionellen Adaptivitätsansätzen nicht mit heutiger Technologie und Materialien erreicht werden können.

Literatur

[HNT+93] Tetsuya Higuchi, Tatsuya Niwa, Toshio Tanaka, Hitoshi Iba, Hugo de Garis und Tatsumi Furuya. Evolving Hardware with Genetic Learning: a First Step Towards Building a Darwin Machine. In *From Animals to Animats*, Seiten 417–424. MIT Press, 1993.

[Kau13] Paul Kaufmann. *Adapting Hardware Systems by Means of Multi-Objective Evolution*. Logos Verlag, Berlin, 2013.

[Koz94] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[MT00] J. Miller und P. Thomson. Cartesian Genetic Programming. In *European Conf. on Genetic Programming (EuroGP)*, Seiten 121–132. Springer, 2000.

Paul Kaufmann studierte Informatik und Mathematik und schloss seine Dissertation auf dem Gebiet der Evolvierbaren Hardware an der Universität Paderborn im Jahr 2013 mit Auszeichnung ab. In den Jahren 2012 und 2013 war er des Weiteren Mitglied des Fraunhofer Institut für Windenergie und Energiesystemtechnik sowie der Gruppe für Energiemanagement und Betrieb Elektrischer Netze der Universität Kassel. Z.Zt. forscht Herr Kaufmann als Postdoktorand am Lehrstuhl für technische Informatik der Universität Paderborn an adaptiven eingebetteten Systemen, multikriteriellen evolutionären Algorithmen, hardwarebasierten Mustererkennungsalgorithmen und Optimierung intelligenter Stromnetze.