Architektur und Entwurfsfluß zur Unterstützung der Anwendungsparallelität durch rekonfigurierbare Rechnersysteme

Sergei Sawitzki

Philips Research Laboratories
Prof. Holstlaan 4 (WDC03)
5656 AA Eindhoven
Niederlande
Sergei.Sawitzki@philips.com

Abstract: Seit den Anfängen des rekonfigurierbaren Rechnens war die Vereinbarung von Prinzipien der parallelen und rekonfigurierbaren Verarbeitung ein wichtiger Forschungsschwerpunkt. Es blieb jedoch fraglich, ob es möglich ist, ein Entwurfsraummodell, eine universelle Architekturvorlage und eine Werkzeugumgebung zur Unterstützung von sowohl Befehls- als auch Datenparallelität auf verschiedenen Granularitätsstufen zu vereinigen. Diese Arbeit stellt die ReSArT Architekturvorlage sowie die DEfInE Entwurfsumgebung vor, womit diese Frage positiv beantwortet wird. Um die Machbarkeit und Lebendigkeit des Konzeptes zu beweisen, wurden verschiedene mit ReSArT und DEfInE erzeugte Architekturinstanzen mit einem Satz von 10 Benchmarks getestet und zeigten für eine prototypische Implementierung bereits vielversprechende Resultate.

1 Motivation und Ziele der Arbeit

Seit der Entwicklung und Einführung der ersten elektronischen Rechenanlagen war die Frage nach Rechenleistung stets eines der wichtigsten System-Bewertungskriterien. Unabhängig von den gegenwärtig erreichten, in Rechenzeit und Durchsatz ausgedrückten, Spitzenwerten bietet der ingenieurtechnische und naturwissenschaftliche Bereich immer noch ausreichend Aufgaben, deren Lösung selbst mit modernsten und schnellsten Rechenanlagen zu aufwendig ist. Überlegungen zu parallelverarbeitenden Rechnern und anderen alternativen Konzepten sind eine logische Konsequenz davon [HP96]. Auch die Einführung der rekonfigurierbaren Komponenten ist ein Schritt in diese Richtung. Neben der reinen Rechenleistung wird der Einsatz der rekonfigurierbaren Logik immer häufiger auch im Zusammenhang mit anderen Aspekten wie Verfügbarkeit und Fehlertoleranz, Reduzierung der Stromaufnahme, Wiederverwendung von Schaltungsentwürfen — um nur einige zu nennen — betrachtet. Die MAKIMOTO-Welle, ein Entwicklungsmodell der Halbleiterindustrie, sieht den Fortschritt in rekonfigurierbarer Technologie als den gegenwärtig bedeutendsten Trend, der den Siegeszug von anwendungsspezifischen in-

tegrierten Schaltkreisen (*application specific integrated circuits*, ASIC) abgelöst hat und noch einige Jahre dominieren wird [Ma00]. Diese Entwicklungen motivieren die Frage, wie die Ansätze und Methoden der Parallelverarbeitung systematisch und vorteilhaft mit den Vorzügen der rekonfigurierbaren Logik kombiniert werden können, auf die diese Arbeit eine Antwort geben soll.

Viele rekonfigurierbare Systeme nutzen auf die eine oder andere Art die explizite oder implizite Anwendungsparallelität und erreichen mitunter für einzelne Anwendungen oder ganze Anwendungsklassen beeindruckende Leistungen [BRV89, GHK+90, At92, BT94, Ol01, Al01]. Der Forschungsbeitrag der vorliegenden Arbeit kann in diesem Umfeld durch folgende Schwerpunkte positioniert werden:

1. Definition eines universellen und parametrisierbaren Entwurfsraummodells für die rekonfigurierbaren Rechnersysteme
2. Definition einer geeigneten Architekturvorlage unter besonderer Berücksichtigung der Parallelitätsunterstützung mit Möglichkeit der Ableitung von Architekturinstanzen
3. Definition einer Entwurfsmethodik, die problemabhängig zu einer geeigneten Architekturinstanz führt und möglichst unabhängig von der Art der Problembeschreibung ist
4. Definition quantitativer Kriterien zur systematischen Untersuchung der Parallelisierungsmöglichkeiten für jede gegebene Anwendung
5. Implementierung geeigneter Werkzeuge zur Evaluation des Entwurfsraummodells, der Architekturvorlage und der Entwurfsmethodik
6. Tests mit einem repräsentativen Satz von Beispielanwendungen
7. Konsequenzen und Rückschlüsse für den Systementwerfer

Der wichtigste Unterschied zu den bisherigen Ansätzen besteht in gleichzeitiger Erfüllung folgender Kriterien:

- eine parametrisierbare, simulations- und synthesefähige Vorlage anstelle einer Speziallösung für ein Problem oder eine Problemklasse
- keine Einschränkung auf Hochsprachen (*high level languages*, HLL) als Ausgangsbasis für die Problembeschreibung
- Unterstützung aller Parallelitätsarten und -ebenen
- keine Einschränkung des maximalen Parallelitätsgrades
- Skalierbarkeit im Bezug auf die Problemgröße und den Parallelitätsgrad

2 Eine Architekturvorlage für die Synthese rekonfigurierbarer Rechnersysteme

Basierend auf dem in der Dissertation beschriebenen Entwurfsraummodell wird die ReSArT Architekturvorlage (Reconfigurable Scalable Architecture Template) definiert (Abbildung 1). Diese besteht aus dem befehlsflußorientierten Rechenkern (*instruction ori-*

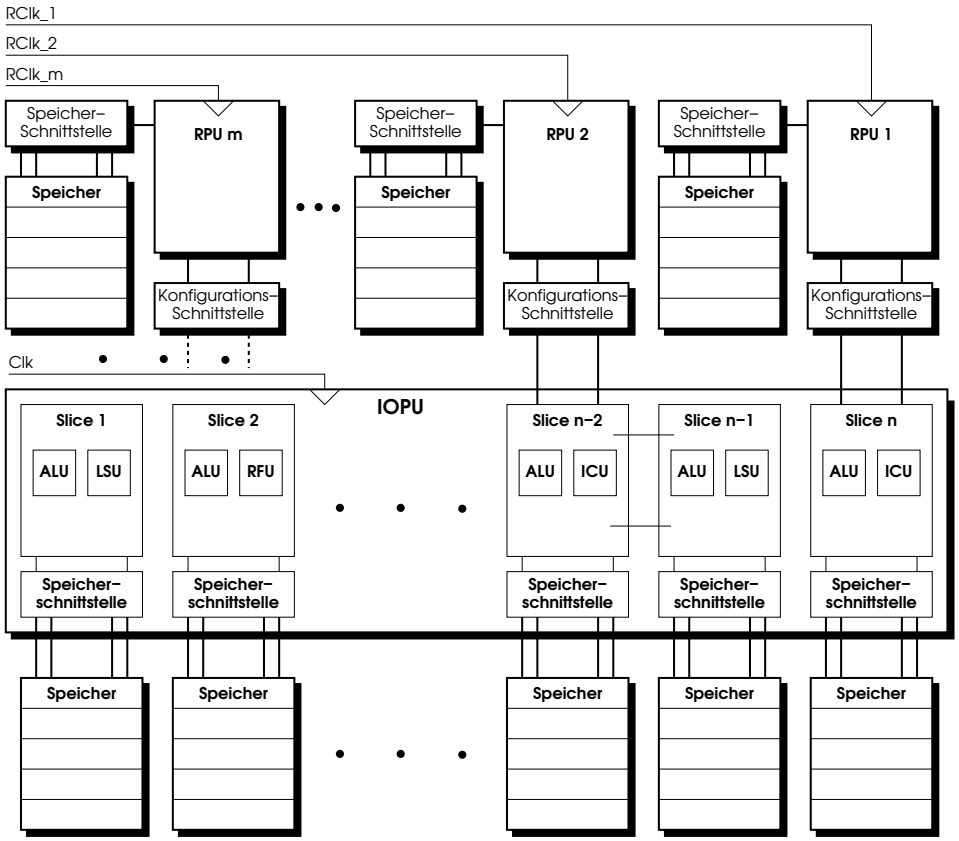

Abbildung 1: Globale Struktur der ReSArT Architekturvorlage

ented processing unit, IOPU) und N_{RPU} datenflußorientierten rekonfigurierbaren Verarbeitungseinheiten (*reconfigurable processing unit*, RPU), wobei $0 \leq N_{RPU}$ gilt. Wie der Name schon andeutet, funktioniert die IOPU nach dem üblichen Prinzip des VON NEUMANN-Rechners: Die einzelnen Rechenschritte werden durch die Befehle gesteuert, die zeitlich sequentiell aus dem Befehlsspeicher (bzw. Befehlscache) geholt werden. Im Gegensatz zu statischen Rechnern erlaubt die IOPU jedoch auch die Integration von rekonfigurierbaren Funktionseinheiten und somit die Definition von Spezialbefehlen, die anwendungsabhängig sind. Die IOPU ist skalierbar aus sogenannten *Slices* aufgebaut, die

jeweils unabhängige oder durch Registerkommunikation verbundene Kontrollfäden einer Anwendung oder auch verschiedene Anwendungen ausführen können. Die rekonfigurierbare Verarbeitungseinheit ist besser für die effiziente Abbildung des Datenflusses rechenintensiver Teile der Anwendung geeignet und ist nur lose mit der IOPU gekoppelt.

Slice ist eine weitestgehend autonome Einheit zur Abarbeitung von RISC-Befehlen, wobei jeweils 2 Befehle parallel abgearbeitet werden. Ein Slice beherbergt eine einfache, vierstufige Pipeline und besteht aus einer Befehls-Ladeeinheit (*Instruction Fetch Unit*, IFU), einem oder zwei Registersätzen, einer Dekodiereinheit (DU) und zwei Verarbeitungseinheiten. Eine Verarbeitungseinheit kann eine ALU (arithmetisch-logisch), eine LSU (Lade-Speicher), eine RFU (rekonfigurierbar, jedoch wesentlich weniger Logikkapazität als die RPU) oder ICU (Schnittstellen-Kontrolle) sein, wobei jeder Slice mindestens eine ALU beinhalten muß. Wenn ein Slice einen Registersatz hat, wird dieser durch die beiden Funktionseinheiten lokal genutzt, bei zwei Registersätzen werden die Registeradressen für andere Slices sichtbar gemacht, so daß eine Kommunikation über die Slice-Grenzen hinweg möglich ist.

Die rekonfigurierbare Verarbeitungseinheit ist ein optionaler Teil der Architekturvorlage, der hauptsächlich zur Implementierung rechenaufwendiger und synchronisationsarmer Anwendungsteile genutzt wird. Aus diesem Grund wird die RPU durch ein vom Systemtakt verschiedenes *RClk*-Signal (RPU-Takt) gespeist und nur mittels loser Kommunikation mit der IOPU synchronisiert. Außerdem besitzt sie eine eigene Speicherschnittstelle, damit der Kommunikationsaufwand weiter reduziert werden kann. Diese Lösung bietet sich insbesondere an, weil die Anforderungen an die Speicherbandbreite und die Taktfrequenz der in der RPU implementierten Schaltung stark anwendungsabhängig sind. Pro Anwendung können drei verschiedene RPU-Varianten generiert werden: Daten-Pipeline, statisch mit gemeinsamer Nutzung der E/A-Ports und dynamisch rekonfigurierbar.

Die RPU basiert auf dem im [BRM99] entwickelten Modell der rekonfigurierbaren Schaltungsstrukturen. Damit kann eine Vielzahl SRAM-basierter, clusterorientierter, inselartiger Mikroarchitekturen von rekonfigurierbaren Ressourcen leicht und flexibel beschrieben werden. Speziell zur Bestimmung des Zeitverhaltens verschiedener konfigurierbarer Zellen und Verdrahtungsstrukturen wurde als Erweiterung des Modells im Rahmen der Dissertation eine Berechnungsvorschrift entwickelt, die ausgehend von den Parametern des einfachsten, aus einer Look-up-Tabelle (LUT) und einem Flipflop bestehenden, Basisblocks die Werte für komplexere Basisblöcke ermittelt.

Vor der Instanzbildung sind die meisten Vorlagenparameter frei. Sie werden erst mit Hilfe der DEfInE Umgebung in mehreren Entwurfsschritten festgelegt.

3 Entwurfsfluß und Entwurfswerkzeuge

Die effektive Nutzung von Rechenressourcen und optimale Abbildung des Problems auf die Architektur können nur durch Einsatz geeigneter Entwurfswerkzeuge, die zusammengefaßt einen teil- oder vollautomatisierten Entwurfsfluß ergeben, erreicht werden. Die ReSArT-Architektur unterstützt verschiedene Parallelitätsebenen. Während zur effektiven

Unterstützung der feinkörnigen Daten- und Befehlsparallelität bekannte Ansätze aus der Compilertechnik genutzt werden können, deren Erweiterungen für den Einsatz auf rekonfigurierbaren Plattformen bereits mehrfach untersucht wurden [AS93, Ka01, BKSS01], bedarf es beim Einsatz der ReSArT-RPU einer besonderen Vorgehensweise, da deren Mikroarchitektur erst durch die Anwendung definiert wird. Dies geschieht mit Hilfe der

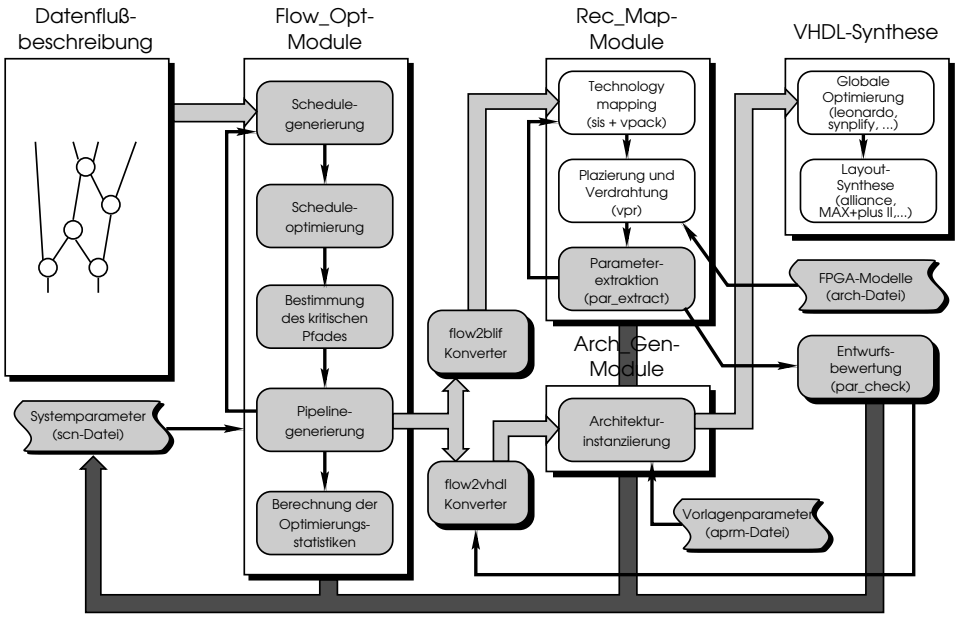

Abbildung 2: DEfInE Entwurfsumgebung und Entwurfsfluß. Die schattierten Komponenten wurden im Rahmen der Dissertation entwickelt.

DEfInE (Design Environment for ReSArT Instance Generation) Entwurfsumgebung (Abbildung 2). Als Teil der Eingabesprache für DEfInE wurde eine spezielle Klasse von gerichteten azyklischen Graphen, die sogenannten $\mathcal{F}$-GAG definiert. Diese zeichnen sich insbesondere dadurch aus, daß sie gleichermaßen zur Beschreibung sowohl fein- als auch grobgranularer Parallelität geeignet sind, keine Einschränkungen bezüglich der Komplexität des Problems bedingen und leicht aus vielen anderen Beschreibungsformalismen erzeugt werden können. Jeder Knoten wird als eine Rechenoperation interpretiert und durch seinen Flächenbedarf, Verzögerungszeit, Anzahl der eingehenden Kanten und die Anzahl der abgehenden Kanten bewertet. Auf der Menge der Kanten ist zusätzlich die Funktion Bit_width definiert, die jeder Kante eine Bitbreite zuordnet. Mit Hilfe einer formalen Transformation können auch einige zyklenbehaftete Graphen in die $\mathcal{F}$-GAG Form überführt und somit dem DEfInE Entwurfsfluß zugänglich gemacht werden.

Der erste Entwurfsschritt wird durch das `flow_opt`-Modul abgedeckt. Dieser besteht aus den Teilschritten Schedule-Generierung, Schedule-Optimierung, Bestimmung des kritischen Pfades, Pipeline-Generierung und Parameterberechnung. Bei der Schedule-Generierung wird jedem Knoten des Graphen eine Stufennummer zugewiesen, wobei der

Initialschedule jedem Knoten die kleinstmögliche Nummer zuordnet. Die Schedule-Optimierung ist eine derartige Umordnung der Knoten im Schedule, daß eines der vier Parameter wahlweise minimiert wird: Maximaler Flächenbedarf einer Stufe, maximale Differenz der Verzögerungszeiten der Knoten innerhalb einer Stufe, maximaler eingehender bzw. abgehender Datenstrom einer Stufe. Diese Optimierungsstrategien sind jeweils auf eine RPU-Implementierungsvariante abgestimmt (die beiden letzten Strategien sind bei der statischen RPU mit gemeinsamer Nutzung der E/A-Ports optimal). Anhand der Berechnung des kritischen Pfades des Schedules werden die Schedule-Stufen auf die Pipeline-Stufen der RPU (oder Teilschaltungen bei dynamischer Rekonfigurierbarkeit) abgebildet. Anschließend werden die Parameter des besten Schedules sowie die relative Veränderung der Parameterwerte bezogen auf den Initialschedule berechnet.

Nach der anschließenden Konvertierung der Schedules in das BLIF-Format [SSL+92] mit dem `flow2blif`-Konverter werden mit Hilfe der `rec_map`-Modul die Schritte *Technology Mapping* sowie Plazierung, Verdrahtung und Extraktion des Zeitverhaltens für rekonfigurierbare Architekturen verschiedener Granularität durchgeführt. Es werden dabei 36 Architekturinstanzen untersucht, die entstehen, wenn man die Eingangsanzahl der LUT von 2 bis 7 und die Anzahl der LUTs pro Cluster von 2 bis 32 (logarithmisch in 2er Potenzen) unabhängig voneinander variiert. Dies geschieht mit Hilfe der SIS, TVpack und VPR Werkzeuge. Nach der Überprüfung der Einhaltung von vorgegebenen Werten für Flächenbedarf, Taktfrequenz sowie Bandbreite des Systems werden die Parameter der Architekturvorlage endgültig festgelegt. Durch `flow2vhdl`-Konverter wird der synthesefähige VHDL-Kode der RPU erzeugt und mit dem durch den `arch_gen`-Modul generierten VHDL-Kode der IOPU zu einer Entwurfsbibliothek zusammengefügt. Anschließend können die Simulation und die Synthese auf Register-Transfer- oder Gatterebene mit den kommerziellen Werkzeugen durchgeführt werden.

4 Zusammenfassung der Ergebnisse

Zum Nachweis der praktischen Implementierbarkeit von ReSArT sowie zum Test der Leistungsfähigkeit der DEfInE Werkzeuge wurden einige Experimente zur Synthese verschiedener Architekturinstanzen auf der Xilinx Virtex II Plattform [Xi01] durchgeführt. Insbesondere sind IOPU-Instanzen für verschiedene Adreßbusbreiten, Verarbeitungsbreiten und Anzahl von Slices sowie verschiedene RPU-Varianten für einen Testsatz von 10 Benchmark-Anwendungen synthetisiert worden. Die Abbildung 3 zeigt die Abhängigkeit des Logikbedarfs und der maximal erreichbaren Taktfrequenz einer IOPU von der Anzahl der Slices. Als Benchmark-Anwendungen für die RPU-Synthese kamen schnelle FOURIER-Transformation, TopHat-Transformation, LPC10 Filter, SISO-Modul eines Turbo-Dekoders, Prozessor zum Vergleich von DNA-Sequenzen, *Game of life* Simulator, Differentialgleichungslöser sowie drei paarweise Kombinationen aus diesen Anwendungen zum Einsatz. Die Abbildung 4 zeigt die relative Veränderung verschiedener Parameter der Schedules für diese Benchmarks nach der Optimierung durch `flow_opt` Modul mit der Verzögerungszeitdifferenz-Optimierungsstrategie. Die erreichten Verbesserungen des entsprechenden Schedule-Parameters im Bereich von 15.38–100% im Vergleich zum In-

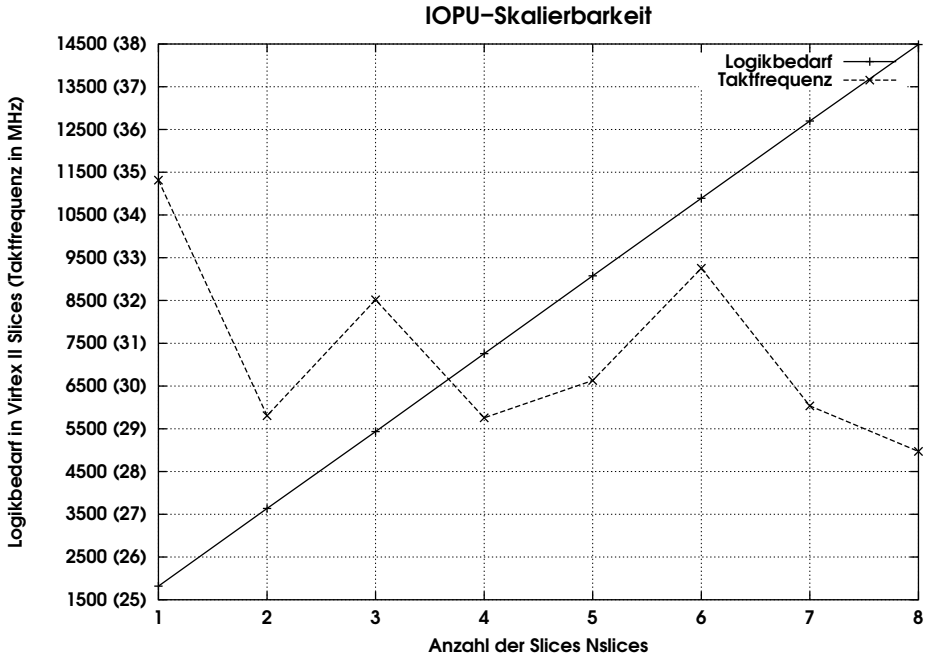

Abbildung 3: Abhängigkeit des Logikbedarfs und der Taktfrequenz von der Anzahl der Slices

itialschedule spiegeln sich nach der Synthese in einer Verbesserung der maximalen Taktfrequenz der RPU im Bereich von 3.2–17.6% (könnte anstelle von Virtex II FPGA die für die jeweilige Anwendung als beste ausgewählte Architekur zum Prototyping genutzt werden, so wäre der ursprüngliche Verbesserungswert weitestgehend erhalten geblieben). Bei anderen Optimierungsstrategien sind ebenfalls Verbesserungen der Ergebnisqualität bezüglich entsprechender Parameter festgestellt worden.

Die Ergebnisse der durchgeführten Experimente können wie folgt zusammengefaßt werden:

- Die ReSArT Architekturvorlage ist leicht skalierbar bezogen auf die Adreßbusbreite, Verarbeitungsbreite und die Anzahl der Slices.
- Ein Virtex II Prototyp der IOPU ist in der Lage, $6 \cdot 10^8$ 16-Bit-Operationen pro Sekunde auszuführen, wobei noch ein wesentlicher Teil der Logik zur Implementierung der RPU frei bleibt.
- Bei den getesteten Benchmark-Algorithmen erreichte die RPU eine maximale Parallelisierungsstufe von 7–30 sowie eine durchschnittliche (d.h. auf die Anzahl der Pipeline-Stufen normierte) Parallelisierungsstufe von 2.92–19.67.
- Die `flow_opt` Optimierungstechniken erlauben eine Erhöhung der Taktfrequenz

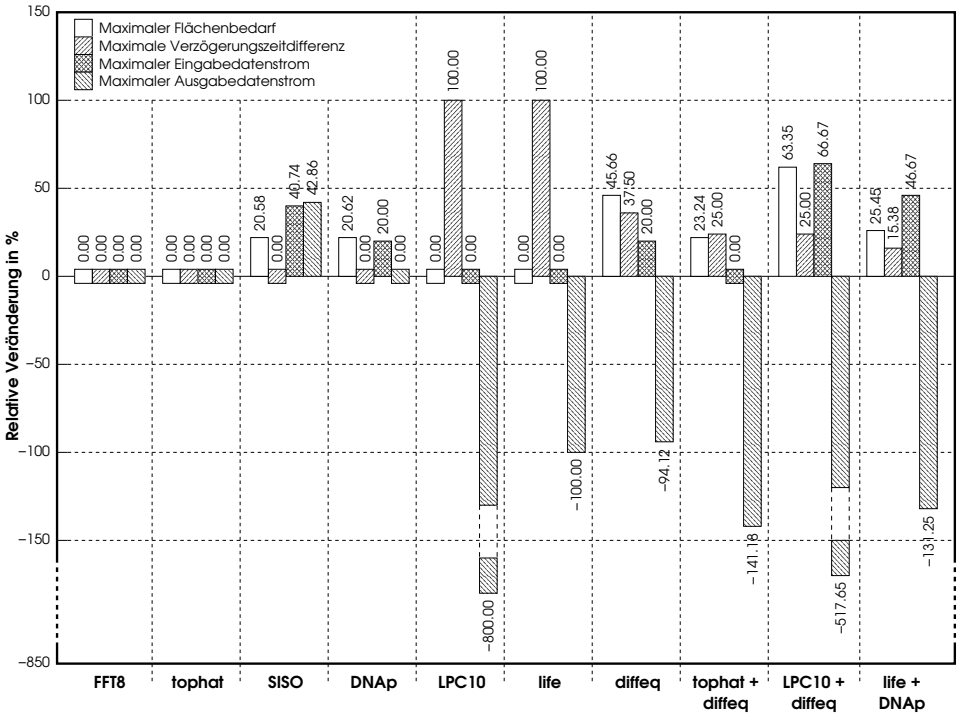

Abbildung 4: Relative Veränderungen der Schedule-Parameter bei der Verzögerungszeitdifferenz-Optimierungsstrategie

um 4.3–17.6% bzw. eine Reduzierung der externen Kommunikationsbandbreite der RPU um 123.4–1417.7%.

- Beschreibung des $\mathcal{F}$-GAG mit Hilfe von größerer Anzahl feingranularer Basisoperationen anstelle von kleinerer Anzahl grobgranularer Basisoperationen führt, bezogen auf die Taktfrequenz und den Durchsatz, zu besseren Ergebnissen.
- Die optimale RPU-Mikroarchitektur hängt nicht nur von der Struktur der Basisoperationen, sondern auch von der Implementierungsvariante der RPU und der Pipeline-Tiefe der RPU ab. Alle getesteten Anwendungen haben ihr Optimum bei LUTs mit 3–6 Eingängen und 1–8 LUT pro Cluster. Clustergrößen von 16 und 32 LUTs zeigen wesentlich schlechteres Zeitverhalten.

Die Experimente mit den Benchmarks zeigen deutlich, daß eine optimale Abbildung der Parallelität einer Anwendung auf die rekonfigurierbare Hardware nicht ohne sorgfältige Untersuchung des Implementierungsraums und quantitative Bewertung jedes einzelnen Entwurfsschritts erfolgen kann. Selbst eine scheinbar unwesentliche Veränderung eines Parameters kann das Ergebnis stark beeinflussen.

5 Abschließende Bemerkungen

Die Arbeit positioniert die Parallelverarbeitung und die Hardware-Rekonfigurierbarkeit im weiten Umfeld der Rechnersystem-Forschung und postuliert, daß diese beiden Ansätze zur Steigerung der Leistungsfähigkeit und der Effizienz von Rechnersystemen vorteilhaft miteinander kombiniert werden können. Der Parallelitätsgrad ist eine stark anwendungsabhängige Größe, viele Algorithmen aus dem ingenieurtechnischen und naturwissenschaftlichen Bereich bieten dennoch mehr Parallelisierungspotential als durch die gängigen statischen Rechnersysteme mit vertretbarem Aufwand unterstützt werden kann. Rekonfigurierbarer Ansatz bietet insofern mehr Flexibilität, als daß die Hardwarestruktur an die spezifischen Anforderungen einer Anwendung angepaßt werden kann, sei es die Wahl der Bitbreite, der Anzahl parallel rechnender Funktionseinheiten oder die Datenpipeline-Tiefe.

ReSArT und DEfInE bilden eine leistungsfähige Ausgangsbasis für die Erforschung der parallelverarbeitenden rekonfigurierbaren Rechnersysteme. Wie bereits mehrfach gezeigt, sind bereits beim heutigen technologischen Stand der rekonfigurierbaren Rechnersysteme beeindruckende Leistungen erreichbar. Die Weiterentwicklung der Halbleitertechnologie wird sicherlich noch höhere Parallelisierungsstufen bei ständig steigendem Durchsatz ermöglichen. Es ist zu erwarten, daß Einchip-Lösungen nach dem hier vorgestellten ReSArT oder ähnlichen Mustern schon bald größere Verbreitung finden, da sie nicht nur rekonfigurierbar, sondern durch das Konzept einer Architekturvorlage auch gut skalierbar sind und somit eine optimale Plattform für den Übergang vom anwendungsspezifischen Schaltkreis zum anwendungsspezifischen System-on-Chip bieten. Diese Arbeit liefert einen Beitrag dazu, diesen Übergang zu erleichtern.

Literatur

[Al01] Altera Corporation: *FFT MegaCore Function User Guide*. March 2001. Version 1.02.

[AS93] Athanas, P. M. und Silverman, H. F.: Processor Reconfiguration through Instruction-Set Metamorphosis. *IEEE Computer*. 26(3):11–18. March 1993.

[At92] Athanas, P. M.: *An Adaptive Machine Architecture and Compiler for Dynamic Processor Reconfiguration*. Phd thesis. Division of Engineering, Brown University. May 1992.

[BKSS01] Braunes, J., Köhler, S., Sawitzki, S., und Spallek, R. G.: Reconfigurable architectures and compilations techniques for exploitation of application parallelism. In: Jähnichen, S. und Zhou, X. (Hrsg.), *Proceedings of the 4th International Workshop on Advanced Parallel Processing Technologies*. S. 242–247. Illmenau. September 17–19 2001.

[BRM99] Betz, V., Rose, J., und Marquardt, A.: *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers. Boston. 1999.

[BRV89] Bertin, P., Roncin, D., und Vuillemin, J.: Introduction to programmable active memories. Research Report No. 3. Digital Equipment Corporation, Paris Research Laboratory. June 1989.

[BT94] Bertin, P. und Touati, H.: PAM programming environment: Practice and expirience. In: Buell, D. A. und Pocek, K. L. (Hrsg.), *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*. S. 133–138. Napa, CA. April 1994.

[GHK+90] Gokhale, M. B., Holmes, B., Kopser, A., Kunze, D., Lopresti, D., Lucas, S. P., Minnich, R. G., und Olsen, P.: SPLASH: A reconfigurable linear logic array. In: Wah, B. W. (Hrsg.), *Proceedings of the 1990 International Conference on Parallel Processing. Volume 1: Architecture*. S. 526–532. Urbana-Champaign, IL. August 1990. Pennsylvania State University Press.

[HP96] Hennessy, J. L. und Patterson, D. A.: *Computer Architecture — A Quantitative Approach*. Morgan Kaufmann Publishers. 2929 Campus Drive, Suite 260, San Mateo, CA 94403, USA. 2nd. 1996.

[Ka01] Kastrup, B.: *Authomatic Synthesis of Reconfigurable Instruction Set Accelerators*. Phd thesis. Technische Universiteit Eindhoven. May 2001.

[Ma00] Makimoto, T.: The rising wave of field programmability. In: Hartenstein, R. W. und Grünbacher, H. (Hrsg.), *Field-Programmable Logic and Applications: the Roadmap to Reconfigurable Computing*. volume 1896 of *Lecture Notes in Computer Science*. S. 1–6. Berlin, Heidelberg, New York. August 2000. Springer-Verlag.

[Ol01] Olay III, R. T.: Xilinx XtremeDSP Initiative Meets the Demand for Extreme Performance and Flexibility. *Xcell*. (40):30–33. July 2001.

[SSL+92] Sentovich, E. M., Singh, K. J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P. R., Brayton, R. K., und Sangiovanni-Vincentelli, A.: SIS: A system for sequential circuit synthesis. Memorandum UCB/ERL M92/41. Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California. Berkeley, CA 94720. 4 May 1992.

[Xi01] Xilinx, Inc.: *Virtex-II 1.5 V Field Programmable Gate Arrays Advance Product Specification*. 25 January 2001. Version 1.3.

Über den Autor

Sergei Sawitzki wurde 1975 in Dresden geboren. Seine Hochschulzugangsberechtigung erlangte er mit einer externen Prüfung zur Eignung ausländischer Studienbewerber am Herder-Institut Leipzig, Abteilung Radebeul. Von Oktober 1992 bis Januar 1998 studierte er Informatik mit Nebenfach Mathematik an der Technischen Universität Dresden. Bereits in seiner Diplomarbeit beschäftigte er sich mit dem Entwurf rekonfigurierbarer Rechner. Nach dem Abschluß als Diplom-Informatiker war er vom April 1998 bis April 2001 Stipendiat des Graduiertenkollegs „Werkzeuge zum effektiven Einsatz paralleler und verteilter Rechnersysteme“ an der Fakultät Informatik der Technischen Universität Dresden. Im Rahmen dieses Graduiertenkollegs entstand seine Dissertation mit dem Titel „Architektur und Entwurfsfluß zur Unterstützung der Anwendungsparallelität durch rekonfigurierbare Rechnersysteme“, mit der er im August 2002 erfolgreich promovierte. Gegenwärtig ist Sergei Sawitzki bei Philips Research in Eindhoven (Niederlande) als Senior Scientist tätig. Seine Forschungsschwerpunkte sind Entwurf eingebetteter Systeme, Reconfigurable Computing und digitale Signalverarbeitung.