Kapitel 1

Einleitung

Diese Arbeit befasst sich mit der Berechnung von Implikationen in booleschen Netzwerken mit programmierbarer Logik - sogenannten Field Programmable Gate Arrays oder kurz FPGAs.

Es wird ein Verfahren vorgestellt, das mit dynamisch programmierbaren FPGAs einen Spezialrechner implementiert, der die Implikationsberechnung wesentlich schneller durchführen kann als ein konventioneller Mikroprozessor.

Das besonderer an dem Verfahren ist, daß die dynamische Konfigurierbarkeit der Bausteine ausgenutzt wird, um die Schaltung des Spezialrechner an die jeweilige Eingabe anzupassen. Das heißt, es wird für jedes zu untersuchende boolesche Netzwerk eine neue FPGA Konfiguration generiert. Dieses Vorgehen wird _Reconfigurable Computing_ genannt und mit RC abgekürzt.

Es ist bekannt, daß es Anwendungen gibt, bei denen Reconfigurable Computing trotzt geringerer Chipfläche einige hundert mal schneller ist als konventionelle Rechnerarchitekturen. Zu den Paradebeispielen gehören Mustererkennung in DNS Sequenzen [Referenz] und Fehlersimulation [Referenz]. Dabei handelt es sich jedoch in der Regel um recht einfache Algorithmen. Zumeist systolische Algorithmen oder Algorithmen mit hoher SIMD

Parallelität. Unklar ist, wie groß der Anwendungsbereich für RC ist. Diese Arbeit möchte zu dieser Diskussion beitragen, in dem sie die Übertragung eines relativ komplexen Algorithmus auf FPGAs untersucht.

Darüber hinaus demonstriert diese Arbeit eine Möglichkeit, die Berechnung von Implikationen erheblich zu Beschleunigen. Implikationsberechnungen haben viele Anwendungen in der Synthese und Verifikation von Schaltungen sowie bei der Testmustergenerierung. Da die Implikationsverechnung sehr viel Rechenzeit benötigt, wird die Berechnung in der der Regel nach einer festgesetzten Zeit abgebrochen. Ein Implikationsbeschleungiger ermöglicht es, in der selben Zeit mehr implikationen zu finden, und führt daher indirekt zu besseren Ergebnissen bei den genannten Anwendungen.

Diese Arbeit gliedert sich in die folgenden Kapitel:

1. Einleitung

2. Reconfigurable Computing
   Einführung in Reconfigurable Computing. Die wichtigsten Ergebnisse und die Problemstellungen der aktuellen Forschung. Diese Kapitel basiert im wesentlichen auf Andre DeHons Doktorarbeit [2].

3. Implikationsberechnung durch Logisches Schließen
   In diesem Kapitel wird das verwendete Algorithmus zur Berechnung von indirekten Implikationen in booleschen Netzen und seine Anwendungen vorgestellt. Grundlage ist das Buch von Professor Kunz . [4]

4. Übertragung des Algorithmus auf FPGAs
   Dynamische Rekonfigurierung, Parallelisierung des Algorithmus, Details des Steuerungsautomatens. Mögliche Erweiterung durch Rekonfiguration während der Laufzeit.

5. Implementierung
   Hier wird detailiert die Schaltung des Implikationsbeschleunigers vor-

gestellt. Es wird auch auf die verwendeten Entwurfmethoden und Software eingegangen.

6. Auswertung
   Mithilfe eines Simulationsprogramms wird das Verhalten des parallelisierten Algorithmus statistisch ausgewertet. Anhand dessen wird eine Formel für die Leistungsfähigkeit des Algorithmus abhängig von Parametern der Implementierung bestimmt. Es werden quantitative Ergebnisse für die Anwendung des Verfahrens auf einige ISCAS Benchmarks vorgestellt.

7. Fazit
   Bewertung der Ergebnisse im Hinblick auf die Fragen: ist das Verfahren praktikabel? Was kann die RC Forschung daraus lernen? Welche Anforderungen an FPGA Architekturen ergeben sich aus dieser Arbeit?

GLOSSAR

RC, Reconfigurable Computing

FPGA

Implikationen

Index: Fehlersimulation, DNS Sequenzen, Reconfigurable Computing, dynamisches Rekonfiguration, Implikationsberechnung, Simulation, ISCAS, FPGA Architektur.

Kapitel 2

Rekonfigurierbare Architekturen

Für die technische Implementierung von Algorithmen wird üblicherweise eines von zwei Verfahren gewählt. Entweder es wird ein Programm für einen Standardprozessor geschrieben oder es wird eine Spezialschaltung verwendet.

Mikroprozessor Im ersten Fall handelt es sich um Standardhardware die im allgemeinen ohne Bezug zu dem zu implementierenden Algorithmus entwickelt wurde. Durch eine Folge von Befehlen die sequentiell abgearbeitet werden lassen sich beliebige Algorithmen implementieren. Es wird eine relativ kleine, zur Chipfertigung festgelegte Hardware mit wenigen Standardoperatoren sehr oft und sehr schnell umkonfiguriert.

Spezialhardware Im zweiten Fall wird der Algorithmus durch eine Schaltung realisiert die speziell für diese Anwendung entwickelt wurde. Die Schaltung kann nur für eine eng begrenzte Klasse von Algorithmen verwendet werden. Die Konfigurierbarkeit beschränkt sich auf wenige Parameter die nur selten geändert werden. Die Realisierung kann als

Anwendungsspezifischer Schaltkreis erfolgen oder als FPGA mit nicht dynamisch veränderbarer Konfiguration.

Die zweite Variante ist in der Regel deutlich Leistungsfähiger, dafür aber unflexibler und teurer sofern nicht sehr hohe Stückzahlen erreicht werden. In erster Näherung kann man sagen, daß Spezialhardware meistens nur dann verwendet wird, wenn die Leistungsfähigkeit von Mikroprozessoren nicht ausreicht.[1]

Der Anteil der Spezialhardware in Computern nimmt jedoch stark zu. Die Chipfläche für 3D Grafik, Klangsynthese, Videodekodierung und ähnliches übersteigt heute in der Regel die Chipfläche für den Mikroprozessor, obwohl diese Eigenschaft in vielen der Bürorechnern nie zum Einsatz kommen, und dort wo sie verwendet werden sind nie alle Einheiten Gelichzeitig aktiv. Wenn man es genau nimmt muß man Multiplizierer und Fließkommaeinheiten auch zur Spezialhardware rechnen, da Fließkommaanwendungen die Außnahme sind. Multiplikationen sind ebenfalls selten [11] und haben fast immer einen Konstanten Faktor [10]. Diese Funktionen sind heutzutage in Prozessoren vorhanden weil wissenschaftliche Anwendungen ein große Rolle bei der Entwicklung neuer Prozessoren spielen, in der Mehrheit der Rechner sind sie aber nur Ballast.

Mikroprozessoren und Spezialhardware stellen in gewisser Hinsicht zwei Extreme dar. Auf der einen Seite der universelle Rechenapparat dessen Funktion mit jedem Takt neu bestimmt wird, und auf der anderen Seite die starre Speziallösung die eventuell unbrauchbar wird sobald sich die Anforderungen ändern.

Zwischen diesen beiden Extremen - Programmierbare Hardware und Hardware mit starrer Funktionalität - gibt es einen großen Raum von architekturellen Alternativen die sich bis jetzt kaum in der Praxis durchsetzen

[1] Bei dieser Diskussion geht es nur um den algorithmischen Teil des Systems. Die Integration von Mikroprozessoren mit Interface-Schaltungen bleibt davon unberührt. Auch in diesem Fall handelt es sich in der Regel um Standardprozessoren.

konnten. Besonders interessant sind in diesem Bereich die Rekonfigurierbaren Architekturen.

Rekonfigurierbare Architekturen stellen eine Alternativ für die extensive Verwendung von Spezialhardware in Standardrechnern dar. Sie ermöglichen es eine komplette Palette spezialisierter VLSI Lösungen die alle gleichzeitig zur Verfügung stehen zu ersetzen durch eine kleine Menge Hardware die jeweils nach Bedarf die gerade geforderte Funktionalität implementiert.

2.1 Charakterisierung rekonfigurierbarer Architekturen

Im allgemeinen nutzen rekonfigurierbare Architekturen einen Großteil ihrer Chipfläche für ein im Vergleich zu Mikroprozessoren, große Zahl an Ausführungseinheiten, die über ein konfigurierbares Verbindungsnetzwerk miteinander verbunden sind. Die Operation jedes Rechenelementes kann ebenso programmiert werden wie die Verbindungen. Verschiedene Teile eines Algorithmus werden über die Chipfläche verteilt und Zwischenergebnisse können direkt von Element zu Element fließen ohne in einem zentralen Speicher abgelegt zu werden. Im einfachsten Fall wird der gesamte Algorithmus in einen Datenflußgraphen überführt und auf die rekonfigurierbare Zielarchitektur abgebildet. Rekonfigurierbare Architekturen tendieren dazu, Aufgaben räumlich zu verteilen, im Gegensatz zu programmierbaren Architekturen - wie Mikroprozessoren - die die Aufgaben zeitlich verteilen.

Die Hauptunterschiede zwischen rekonfigurierbaren Architekturen und konventionellen Prozessoren sind:

Befehlsverteilung Die Konfigurationen werden nicht Takt für Takt aus einem Zentralen Befehlsstrom an die Ausführungseinheiten verteilt, sondern werden lokal gespeichert. Dadurch können wesentlich größere und komplexere Konfigurationen verwendet werden. Da die externe Bandbreite natürlich weiterhin begrenzt bleibt, folgt daraus, daß Rekonfigurierbare Architekturen nur selten neue Konfigurationen aus externem Speicher laden. Intern können aber mehrere Konfigurationen gespeichert sein. Siehe z.B.[14]

Datenfluß für Zwischenergebnisse Zwischenergebnisse werden parallel von der Quelle zur Senke geleitet anstatt sämtliche Kommunikationen über einen zentralen Ressource durchzuführen, wie zum Beispiel eine Registerbank.

Mehr Ausführungseinheiten, feinere Granularität Es gibt viele Ausführungseinheiten die unabhängig voneinander konfiguriert werden können. Dadurch können mehr und flexiblere Operationen ausgeführt werden als bei einem konventionellen Prozessor.

Verteilte Ressourcen vermeiden Flaschenhälse Speicher, Verbindungsnetzwerk und Ausführungseinheiten sind nicht in großen Blöcken angeordnet, sondern über den Chip verteilt und können bei Bedarf bestimmten Aufgaben zugeordnet werden. Dadurch kann Parallelität und Lokalität des Algorithmus ausgenutzt werden.

Viele dieser Eigenschaften wurden erst durch die Verfügbarkeit sehr hoch integrierter Schaltkreise ermöglicht. Dies erklärt, wieso diese Architekturen erst seit relativ kurzer Zeit systematisch erforscht werden.

Die oben angegebene unformale Charakterisierung deckt einen recht großen Entwurfsraum ab. Deshalb unterscheiden sich die Veröffentlichten Ansätze sehr stark:

Konfigurierbare Datenpfade Eine Reihe von Ausführungseineinheiten wird über ein konfigurierbares Datenflußnetzwerk verbunden und dann mit einem programmierbaren Datenstrom gefüttert. Die Granularität ist oft im Bereich der Mikroprozessoren, wie zum Beispiel beim XPUTER.[13].

Prozessoren mit konfigurierbaren Ausführungseinheiten Ein konventioneller Mikroprozessor bei dem die Funktionalität der Prozes-

sorbefehle über FPGA ähnliche Strukturen frei konfiguriert werden kan. Beispiele: GARP[15].

Felder prozessorähnlicher Ausführungseinheiten PADDI [17] ist beinahe ein extremes Beispiel einer konventionellen MIMD Architektur. Viele vollwertige 8-Bit Prozessoren sind dabei über ein Verbindungsnetzwerk verbunden. MATRIX [16] hingegen besteht aus vielen 8 Bit ALUs mit jeweils 256 Byte Speicher und einen hierarchischen Verbindungsnetzwerk. Die Einheiten können unter anderem als Programmspeicher mit Befehlszähler, als ALU in einem Prozessor oder als Knoten in einem Datenflußnetzwerk verwendet werden.

Zelluläre Automaten Dies ist das älteste Beispiel einer konfigurierbaren Architektur. Sie wurden erstmals von von Neumann 1966 beschrieben und ähneln schon sehr stark den FPGAs.[18].

FPGA basierende Architekturen Mit FPGAs lassen sich weitgehend beliebige Schaltungen realisieren, sie lassen sich also auch als Rekonfigurierbare Rechner nutzten. Ihr Nachteil ist der hohe Flächenbedarf für die sehr detaillierte Konfiguration, sie sind jedoch die flexibelste aller hier vorgestellten Architekturen und sind als als Standardprodukte kommerziell verfügbar.

2.2 FPGA basierende Rechner

Die meisten Forschungsprojekte zu rekonfigurierbaren Architekturen beschäftigen sich mit FPGAs. Die relativ große Zahl der FPGA basierenden Projekte erklärt sich hauptsächlich daraus, daß FPGAs die einzige rekonfigurierbaren Architekturen sind, die als Standardbaustein im Handel erhältlich sind. Einige RA Gruppen beschäftigen sich aber auch gezielt mit FPGAs, da sie die meisten Freiheitsgrade und damit den größten Raum für Experimente. Zum Beispiel lassen sich alle anderen Architekturen - im

allgemeinen und Flächen und Geschwindigkeitseinbußen - auf FPGAs abbilden. Weiterhin sind FPGAs mit ihrer extrem feinen Granularität gerade in den Anwendungen gut, in denen die heutigen grob granularen Hochleistungsprozessoren besonders schlecht abschneiden.

2.2.1 Erfolge FPGA basierender RA

Andre DeHon führt in [2] Beispiele für die Rechenleistung der bekanntesten FPGA Rechner auf:

PAM DEC PRLs Programmable Active Memory Projekt verwendet Felder von Xilinx FPGAs. Perle-1 bestand 1992 aus 23 XC3090 auf einer Einsteckkarte für eine Workstation. Als Leistung geben die Autoren von [19] die folgenden Beispiele an:

- Multiplikation großer Zahlen 16x schneller als Cray-II
- Mit 600kBit/s, 512-Bit RSA Dekodierung. Schnellste Implementation zum Zeitpunkt der Veröffentlichung
- Suche in Zeichenketten mit ähnlicher Leistung wie eine Speziallösung aus 28 VLSI Schaltkreisen.
- Diskrete Cosinustransformation mit 15 GOPS.

SPLASH SPLASH von SRC besteht aus 32 XC3090. SPLASH führt laut [20] DNS Sequenzierung 300 mal schneller aus als ein CRAY-II

Logikemulation FPGA basierende Spezialrechner zur Logikemulation waren wahrscheinlich die erste kommerzielle Anwendung rekonfigurierbarer Architekturen. Der erste FPGA basierende Emulator war der Realizer. Er bestand aus 42 XC3090 und 160 XC2018 für die Kommunikation. Der Realizer konnte Schaltungen mit etwa 10000 Gatteräquivalente mit eine Geschwindigkeit von einigen MHz simulieren

Fehlersimulation Der Rechenzeitaufwand und die gute Parallelisierbarkeit einiger Fehlersimulationsalgorithmen hat Fehlersimulation zu einer weiteren kommerziellen Anwendung für FPGA basierende Rechner werden lassen.

Es hat sich gezeigt daß FPGAs überall dort gut sind wo auf kleinen Datenworten operiert wird, wie zum Beispiel beim DNS-Sequenzieren, Fehlersimulation und Logikemulation, und bei Anwendungen die sich nicht gut auf die Operationen eines Standardprozessors Abbilden lassen, wie beispielweise RSA Dekodierung. Der Hauptvorteil, der Rekonfigurierbare Architekturen erfolgreich mit VLSI Spezialschaltungen konkurrieren läßt, ist daß sich die Schaltungen dynamisch an die Eingabendaten anpassen lassen. Dies bringt bei der Mustererkennung (DNS Sequenzierung und Suche in Zeichenketten) und beim RSA dekodierung den eigentlichen Leistungssprung. Dynamische Optimierungstechniken werden auch für Mikroprozessoren untersucht. Die erste kommerzielle Anwendung dürfe SUNs Hot Spot Compiler sein [24].

DieEignung von FPGAs ist bisher überwiegend für systolische Algorithmen untersucht worden. Diese Arbeit soll an einem Beispiel untersuchen, wie gut FPGAs für einen etwas komplizierteren, rekursiven Algorithmus auf eher zufälligen Graphen abschneiden.

2.2.2 Neue FPGA Architekturen

Die auf dem Markt verfügbaren FPGAs wurden nicht im Hinblick auf Rekonfigurierbare Architekturen entwickelt. Deshalb stellt sich die Frage, wie soll ein FPGA aussehen der für RC geeignet ist? Unter anderem ergeben sich die folgenden Anforderungen:

Einfaches Timingmodell (BRASS, TSFPGA, DPGA) Synchrone Arbeitsweise mit einer garantierten Taktfrequenz unabhängig vom Fanout eines Signals und der detaillierten Plazierung sind erforderlich,

um die Algorithmen für die dynamische Schaltkreissynthese einfach und schnell zu halten. Außerdem würde sonnst die Taktfrequenz zu stark von den Eingabedaten abhängen.

Pipelining (BRAS, TSFPGA) Die Ausführungseinheiten eines FPGAs sind typischerweise sehr schnell im Vergleich zum Verdrahtungsnetzwerk. Da bei vielen Algorithmen der Datendurchsatz wichtiger ist, als die Verzögerungszeit lohnt es sich Flip-Flops in das Verbindungsnetzwerk einzufügen. Da es dann im allgemeinen Verbindungswege gibt die eine unterschiedlich viele Takte benötigen, sind dann zusätzlich Retiming Register an den Eingängen der Ausführungseinheiten nötig. BRASS Trumpet kann auf diese Weise für beliebige Anwendungen Taktraten von 350 MHz garantieren, was in herkömmlichen FPGAs nur in Ausnahmefällen möglich ist.

Schnelle Neukonfiguration (Xilinx 6200, BRASS, DPGA) Wenn sich die Konfiguration des FPGA im Betrieb häufig ändert, sollten das ändern der Konfiguration schnell gehen. Verschiedene Techniken können dabei helfen:

- Breiter, schneller Datenpfad zur Neukonfiguration (XC6200, BRASS)
- Änderung von Teilkonfiguration (XC6200, BRASS)
- (DRAM) Konfigurationscache mit breitem Datenpfad zum Konfigurationsspeicher (neoRAM [22], BRASS, DPGA)
- Double Buffering, eine zweite Konfiguration läßt sich verändern während die erste Konfiguration benutzt wird. Die Umschaltung erfolgt in einem einzelnen Takt. (DPGA)
- Mehr als eine aktive Konfiguration. Beim DPGA existieren mehrere Konfigurationen. Die Schaltung kann jeden Takt zu einer neuen Konfiguration wechseln. Dadurch lassen sich unter anderem endliche Automaten effizient realisieren.

Hierarchisches Verbindungsnetzwerk (BRAS, DPGA, TSFPGA) RA erfordern sehr große FPGAs. Hierarchische Netzwerke skalieren besser und erfordern weniger Schaltelemente und weniger Konfigurationsspeicher. Eine Veröffentlichung von Leiserson hat die Frage aufgeworfen ob vielleicht sogar eine lineare Anzahl von Schaltern ausreicht [23]. Außerdem vereinfachen hierarchische Netzwerke die Plazierungsalgorithmen. Im Extremfall ist nur eine Partitionierung des Schaltnetzes erforderlich.

2.2.3 Vorteile von FPGA Rechnern

Die realisierung einer Schltung als FPGA ist relativ ineffizient. Für die Konfiguration, vor allem aber für die flexible Verdrahtung wird sehr viel Fläche benötigt. Eine typische Aufteilung der Chipfläche eines FPGAs ist 90% für die Verdrahtung, 9% für den Konfigurationsspeicher und 1% aktive Logik. Dies erhöht die Kosten und reduziert die Geschwindigkeit einer Schaltung gegenüber einer VLSI Realisierung.

Dennoch können Rekonfigurierbare Rechner auf FPGA Basis im Vergleich zu konventionellen Architekturen sehr effizient sein. Das kommt daher, daß auch bei Mikroprozessoren ein großer Teil der Hardware nur selten oder n icht effizient genutzt wird. In Verbindung mit der oft höheren Rechenleistungen von RAs führt dies dazu, daß die Rechenleistung realtiv zur Chipfläche bei Rekonfigurierbaren Architekturen im allgemeinen besser ist als bei Standardprozessoren. Die Fläche wird für dieses Effizienzmaß sinnvollerweise auf λ, die halbe Strukturgröß des Herstellungsprozesses, normiert.Andre DeHon gibt in [2] einen quantitativen Überblick über die Effizienz diverser Rechnerkonzepte.

2.3 RA im Vergleich zu Standardprozessoren

Mikroprozessoren haben sich in den vergangenen Jahrzehnten durchgesetzt, da sich durch ihre Flexibilität die Entwicklungskosten auf eine große Zahl

von Anwendungen umlegen läßt. Durch ihre freie Programmierbarkeit kann praktisch mit einem beliebigen Prozessor jedes berechenbare Problem gelöst werden. Die Wahl des Prozessor wird nur bestimmt durch einen Abwägung von Verarbeitungsgeschwindigkeit und Preis sowie unter Umständendurch weitere Kriterien wie Stromverbrauch oder Platzbedarf.

Die Architektur der Prozessoren ist dabei traditionell dadurch bestimmt, daß eine kleine Anzahl aktiver Ressourcen ständig wieder verwendet wird. In den Anfängen der Computerentwicklung war dies notwendig, da Hardware teuer und fehleranfällig war. Schon seit langem werden jedoch komplette Prozessoren in einem einzelnen Baustein realisiert, und die verfügbare Fläche wächst ständig weiter. Doch wie läßt sich diese Fläche nutzen?

2.3.1 Probleme hoher Instruktionsraten

Die konventionellen Prozessorarchitekturen können diese zusätzliche Fläche nicht besonders gut nutzen. Ein gewöhnlicher Prozessor besitzt im allgemeinen einen Datenpfad der in jedem Arbeitsschritt durch einen Befehl neu konfiguriert wird. Um den Platzbedarf für Programme nicht zu groß werden zu lassen, und um schnell genug neue Konfigurationen nachladen zu können, wird die Menge der möglichen Konfigurationen klein gehalten. In der Regel sind überwiegend einfache logische und arithmetische Operationen wie Addition, Subtraktion oder Vergleiche möglich. Operationen die nicht im Befehlssatz enthalten sind müssen durch eine längere Befehlsfolge realisiert werden.

Die Leistungsfähigkeit der Prozessoren wird schon lange nicht mehr von der für die Berechnung der Operationen benötigten Zeit bestimmt. So benötigt beispielsweise der in Hewlett Packards PA-RISC 8200 Prozessor verwendete 64-Bit Addierer, in aus heutiger Sicht schon fast altmodischer 0,5 μm Technologie, nur 0,7 ns für eine Addition.[9] Die Zykluszeit des Prozessor beträgt jedoch 5ns. Das bedeutet, es hat wenig Sinn, die zusätzliche verfügbare Hardware zu verwenden um die Ausführung der Standardoperationen

zu beschleunigen.

Die Leistung der heutigen Prozessoren wird stattdessen maßgeblich dadurch bestimmt, wie schnell der Datenpfad umkonfiguriert werden kann, das heißt, wie schnell neue Befehle geladen und dekodiert werden können. Negativ auf die Leistung wirken sich dabei langsame Zugriffzeiten auf den Hauptspeicher und Datenabhängigkeiten bei Verzweigungen des Befehlsstroms. Um trotz dieser Schwierigkeiten ausreichend schnell Konfigurationen laden zu können, wird ein erheblicher Aufwand betrieben. Befehlscache, Sprungzielcaches, Sprungvorhersage, Spekulative Ausführung und verzögerte Verzweigungen gehören in diese Kategorie. Diese Prozessorkomponenten benötigen oft einen weit größeren Teil der Chipfläche als die Recheneinheiten die die Eigentliche Arbeit machen sollten.

Eine Möglichkeit die Leistung zu erhöhen ist, mehrere Befehle gleichzeitig auszuführen. Werden s Befehle gleichzeitig ausgeführt, so sind s Ausführungseinheiten gleichzeitig aktiv. Es wird also absolut gemessen mehr Hardware sinnvoll genutzt. Die Effizienz sinkt dadurch jedoch weiter,

Um auf diese Weise die s-fache Befehlsrate zu erreichen steigt nämlich der Flächenbedarf für Befehlscaches mindestens proportional zu s. Auch der Aufwand für die anderen im letzten Absatz genannten Maßnahmen steigt.

Wenn dabei das sequentielle Prozessormodell beibehalten werden soll müssen zusätzlich Abhängigkeiten zwischen den gleichzeitig ausgeführten Befehlen aufgelöst werden. insgesamt steigt der Aufwand für die Steuerlogik bei superskalaren Prozessoren schneller als linear mit der Zahl der parallel ausgeführten Befehle, so daß der Anteil der für die Ausführungseinheiten verwendeten Chipfläche tendenziell sinkt.

Dieses Problem ist lange bekannt, und es gibt mehrere Ansätze den Befehlsstrom zu entlasten indem mit einer Datepfadkonfiguration mehrere Operationen ausgeführt werden.

Vektorrechner Vektorrechner führen die selbe Operation auf einem Vektor statt auf einem Skalar aus. Die Konfiguration des Datenpfades bleibt dabei über eine gewisse Anzahl von Takten bestehen.

MMX Die Multimedia Erweiterungen verschiedener Chiphersteller sind ein Spezialfall der Vektorrechner bei dem durch minimale Eingriffe in die Architektur eines Skalar- oder Superskalarprozessors Vektoren geringer Länge - typischerweise 2 bis 8 Zahlen - bearbeitet werden können.

VLIW (Very Large Instruction Word) Prozessoren mit einem sehr großen Befehlswort reduzieren zwar im allgemeinen nicht die Anforderungen an die Befehlsbandbreite, die Steuerlogik wird jedoch dadurch vereinfacht, daß es per definition keine Abhängigkeiten zwischen gleichzeitig ausgeführten Operationen gibt, und daß alle Operationen eines Befehls gleichzeitig ausgeführt werden können. Es können also bei gleichem Aufwand für die Steuerlogik mehr Operationen gleichzeitig ausgeführt werden.

Rekonfigurierbare Architekturen versuchen möglichst ganze Teilalgorithmen in einer Konfiguration unterzubringen und die Rekonfigurationsrate so weit zu reduzieren, daß der Nachteil der erheblich größeren Konfiguration ausgeglichen wird.

2.3.2 Unangepaßte Wortbreite

Aus den oben angegebenen Gründen kann die Wortbreite der Prozessoren erhöht werden ohne die Fläche oder die Geschwindikkeit des Prozessors wesentlich zu beeinflussen. Die Wortbreite wächst daher tendenziell in der Hoffnung, mehr Arbeit mit einem Befehl erledigen zu können. Dies Erhöht zwar die absolute Leistung, reduziert aber die Effizienz, da die Fläche der Datencaches proportional zur Wortbreite des Prozessors steigt, die Leistung bei den meisten Anwendungen aber nur geringfügig steigt.

Prozssoren sind also bei Anwendungen die auf kleinen Datenworten operieren sehr innefizient (DNS Sequenzierung, RSA Dekodierung). FPGAs können sich hingen optimal der von der Anwendunge benötigten Wortbreite anpassen. Trotzdem sind FPGAs bei Prozessortypischen Wortbreiten

ineffizient, da die Fähigkeit jedes Bit einzeln zu konfigurieren für diese Anwendungen nicht genutzt wird. Es gibt daher eine Reihe Rekonfigurierbarer Architekturen die mit mittleren Wortbreiten von etwa 8 Bit arbeiten.

Der in dieser Arbeit vorgestellt Algorithmus arbeitet mit 4-Wertiger Logik. Bei einer Implementierung auf einem 64-Bit RISC prozessor hat man die Wahl entweder immer nur 2 Bit - etwa 3% - des Datenpfades und des Datencaches zu benutzen, oder mehrere Signalwerte in ein Wort zu packen, was Zusatzaufwand für die Dekodierung bedeutet.

2.3.3 Schlechte Ausnutzung von Konstanten

Aus Sicht der Ausführungseinheit eines Prozssors sind in der Regel alle Operanden Variablen. In den meisten Anwendungen bleiben viele Operanden jedoch über längere Zeit konstant. Nicht veröffentlichte Ergebnisse der BRASS Gruppe in Berkeley zeigen daß bei einigen SPECint Anwendungen mehr als 80% der Variablen Bit höchstens einmal während der Programmausführung geändert werden.

Es gibt in der Compilerforschung Ansätze, selbstmodifizierende Software zu generieren, die diesen Effekt zur Softwareoptimmierung nutz. Rekonfigurierbare Architekturen profitieren jedoch noch weitaus stärker davon, da sie nicht nur die Zahl der Operationen sondern auch die Komplexität der Operatoren reduzieren kann. Beim DNS Sequenzieren Beispielsweise reduziert sich die Fläche durch die Spezialisierung auf eine zu suchende Sequenz um den Faktor 2. Konstantenmultiplikationen mit einer n-Bit Konstanten erfordern im schlimmsten Fall $n/2$, erwartet etwa $\sqrt{n}$ Additionen. Wenn, wie bei FIR Filtern, die Quantisierung der Koeffizienten beeinflußt werden kann, kann die Zahl der Additionen noch weiter reduziert werden.

2.3.4 Schlechte Verwaltung von Zwischenergebnissen

Um ein Zwischenergebnis, das nicht in den Registern abgelegt werden kann, zu verwenden, muß ein Prozessor einen hohen Aufwand betreiben. Effektive

Addressen müssen berechnet werden, Cachetags für mehrere Cahces verglichen werden, virtuelle Addressem müssen in physikalische Addressen umgerechnet werden, und so weite Die dafür verwendete Hardware betreibt immer den gleichen Aufwand, auch wenn die Zugriffsfolge vorhersagbar ist.

In Rekonfigurierbaren Architekturen können die Speicherzugriffe wesentlich besser optimiert werden. Reine Zwischenergebnisse müssen oft gar nicht gespeichert werden, sondern können gleich an die nächste Ausführungseinheit weitergegeben werden. Außerdem kann die Speicherarchitektur in der Regel an die Anwendung angepaßt werden. Die in dieser Arbeit vorgestellte Schaltung kann zum Beipspiel in einem Takt etwa 250 Byte Registerinhalte mit Daten aus einem Stack austausche. Bei 10 MHz entspricht das einer Bandbreite von 5 GByte/s. Parallel dazu kann etwa die doppelte Datenmenge zwischen den Ausführungseinheiten bewegt werden. Durch die direkte Weiterleitung der Zwischenergebnisse an die nächste Ausführungseinheit entfallen außerdem die sonst für Graphen erforderlichen Zeigerdereferenzierungen

2.4 RAs fordern hohen Entwicklungsaufwand

Das größte Hindernis für die Verbreitung Rekonfigurierbarer Architekturen ist das fehlen asugereifter Entwicklungssoftware. Die Automatische Synthese von Algortihmen für Rekonfigurierbare Hardware ist ein noch nicht sehr weit fortgeschrittenes Gebiet aktueller Forschung. Besonders für den wichtigen Aspekt der dynamischen Rekonfigurierung gibt es kein ausgereiftes Entwurfswerkzeug. Die dynamische Spezialisierung des Programmes auf lokale Konstanten steckt auch für normale Prozessoren noch in den Kinderschuhen.

Das führt dazu, daß der Aufwand für die Implementierung einges Algorithmus auf einer RA noch fast so hoch ist wie für die Entwicklung von Spezialhardware. Dadurch können RA trotz ihrer theoretischen Eignung für allgemeine Aufgaben in der Praxis heute nur für Spezialanwendungen

eingesetzt werden.

Die Entwicklung von Compilern für diesen Bereich ist sehr schwierig. Deshalb versucht das BRASS Projekt der Uni Berkeley die FPGA Achitektur so zu optimieren, daß die Leistung nicht ganz so stark von der Qualität der Synthesesoftware abhängt. Dazu dienen die in Kapitel 2.2.2 vorgestellten Techniken.

BELEGEN: Chipfläche für ALU vs REST

BELEGEN: Prozentsatz der Anwendungen mit FPU, MUL, MMX, etc.

INDEX: Mikroprozessor, Prozessor, superskalar, Sprungvorhersage, spekulative ausführung

Kapitel 3

Implikationsberechnung in booleschen Netzwerken

TODO: Andre mit Accent

Fehlerkorrektur: s/ß, c statt k (active), c statt z (processor), Klein/groß, Anglizismen, fehlende Buchstaben am Wortende (besonders e)

Literaturverzeichnis

[1] Michael Chu, Kolja Sulimma, Nick Weaver, Andre DeHon, John Wawrzynek. „Object Oriented Circuit-Generators in Java“ in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machins, 1998.*

[2] Andre DeHon. „Reconfigurable Architectures for General-Purpose Computing“. A.I. Technical Report No. 1586, Massachusets Institute of Technology, Artificial Intelligence Laboratory, 1996

[3] Andre DeHon, John Wawrzynek. „The Case for Reconfigurable Processors“. University of California at Berkeley, Computer Science Division, 1997

[4] Wolfgang Kunz, Dominik Stoffel. „Reasoning in Boolean Networks“, Kluwer Academic Publishers, 1997.

[5] Tafertshofer. ICCAD'98

[6] Xilinx, „The Programmable Logic Data Book 1998“, 1998

[7] Xilinx, „Xilinx Netlist Format (XNF) Specification“, version 6.1, 1995

[8] Peixing Zhong, Margeret Martonosi, Pranav Ashar, Sharad Malik. „Accelerating Boolean Satisfiability with Configurable Hardware“ in *FPGA '98*

[9] HP, Addiererdesign für den HP 8200

[10] HP, Strength Reduction

[11] Patterson Quantitative

[12] Pak K. Chan and Martine D. F. Schlag, „Acceleration of an FPGA router“, Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[13] Hartenstein?

[14] DPGA

[15] GARP

[16] Matrix

[17] PADDI

[18] vonNeumann zellular

[19] Patrice Bertin, Didier Roncin and Jean Vuillemin. Introduction to Programmable Active memories. PRL Report 3, DEC Paris Research Laboratory, Juni 1989

[20] Maya Gokhale, William Holmes, Andrew Kopser, Sara Lucas, Ronald Minnich, Douglas Sweely and Daniel Lopresti. Building and Using a Highly Programmable Logic Array. IEEE Computer, 24(1):81-89, Januar 1991

[21] Joseph Varghese, Michael Butts and Jon Batcheller. An Efficient Logic emulation System. IEEE Transaction on Very Large Scale Integration Systems, 1(2):171-174, Juni 1993

[22] Self-Modifying, On-the-Fly-Reconfigurable Logic, M. Chatter, Reconfigurable Architectures - High Performance by Configware, ITpress Verlag, herausgegeben vonReiner W. Hartenstein und Viktor K. Prasanna, 1997

[23] Leiserson, hierechical routing on FAT trees

[24] SUN, HotSpot