Hochleistungsrechner: Aktuelle Trends und Entwicklungen
Wintersemester 2016/17

FPGA für HPC

Konrad Pröll
Technische Universität München

02.02.2017

Zusammenfassung

In dieser Seminararbeit wird zuerst der Aufbau eines Field Programmable Gate Array - FPGA - vorgestellt. Anschließend wird, nach einer Übersicht über die Programmierung von FPGAs, deren Anwendung im Hochleistungsrechnen aufgezeigt sowie deren Stärken und Schwächen hierbei beurteilt. Zum Schluss soll das Potenzial von FPGAs für Zukunft beurteilt werden.

1 Einführung zu FPGAs

Seit Xilinx 1985 mit dem *XC2064* das erste Field Programmable Gate Array, kurz FPGA, auf den Markt brachte, hat sich diese Technik stetig weiterentwickelt und zu Beginn dieses Jahrtausends sogar den Zugang zum Hochleistungsrechnen gefunden.
Ein FPGA setzt sich von gewöhnlichen integrierten Schaltkreisen dadurch ab, dass sich der Begriff *programmable* hierbei nicht auf normale Programmierung bezieht, sondern darauf, dass der Aufbau der logischen Schaltung programmiert werden kann. Man spricht hierbei davon, dass das FPGA *konfiguriert* wird. Dabei sind nicht nur die einzelnen Logikgatter konfigurierbar, sondern, da diese auch untereinander verbunden sind, auch größere Schaltungen realisierbar
Während der XC2064 noch über lediglich 64 Logikblöcke verfügte, wobei jeder davon zwei Look-Up-Tabellen für drei Inputs hatte, und über 38 Pins In- bzw. Output verarbeiten konnte, haben modernere Chips wie etwa der *Xilinx XCVU440* über 4,4 Millionen Logikblöcke und Look-Up-Tabellen mit 6 Inputs sowie 1456 Pins.
Im Folgenden soll ein genereller Einblick in Aufbau und Programmierung von FPGAs gegeben werden, bevor deren Anwendung im HPC beleuchtet wird. Abschließend soll ein Ausblick auf zukünftiges Potenzial der Technologie gegeben werden.

Abbildung 1: Der Xilinx XC2064. [1]

2 Aufbau von FPGAs

Eine FPGA setzt sich zuerst aus mehreren CLBs (konfigurierbare Logikblöcke), welche untereinander verbunden sind, zusammen. CLBs sind hierbei einzelne Gatter, die je nach Aufbau unterschiedlich mächtig bzw. performant sind. Zuerst soll hierbei in 2.1 der Aufbau eines Logikblocks vorgestellt werden werden. In 2.2 soll danach der Aufbau eines aus mehreren CLBs bestehenden FPGAs vorgestellt werden.

2.1 Aufbau eines Logikblocks

Ein Logikblock besteht grundstzlich aus drei verschiedenen Komponenten: Look-Up-Tabellen, Flip-Flops und Multiplexern. Diese sollen nun vorgestellt werden. Anschließend soll dargestellt werden, wie sich ein Logikblock aus diesen Komponenten zusammensetzen kann.

2.1.1 Look-Up-Tabellen

Ein wichtiges Konzept beim Design von Logikblöcken sind Look-Up-Tabellen. Diese können vom Nutzer beschrieben werden, aber teilweise auch durch die auf dem FPGA ausgeführte Software selbst. Deren Inhalt wird im RAM gespeichert. (vgl 5) Bei einer 3-input Tabelle würde die Konfiguration für

$$\text{Output} = (\text{A} \wedge \text{B}) \vee \text{C}$$

aussehen:

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tabelle 1: Look-Up-Tabelle für $(\text{A} \wedge \text{B}) \vee \text{C}$

Look-Up-Tabellen kommen in beinahe allen Logikblöcken vor. Meist wird dabei die Anzahl der maximal möglichen Eingabeparameter spezifiziert. Über eine n-Input LUT können logische Verknüpfungen mit n Parametern dargestellt werden. [17]

2.1.2 Flip-Flop

Ein Flip-Flop ist eine Schaltung, die nichtflchtig ein Bit speichern kann. Sie hat im Regelfall ein Eingangssignal, um das Bit auf 1 zu setzen, ein Reset-Signal, mit dem das Bit auf 0 gesetzt wird, sowie ein Ausgangssignal sowie dessen Invertierung. Bei FPGAs können Flip-Flops als Buffer genutzt werden. Ein Bit kann dort über Taktzyklen hinweg gespeichert und später wieder verwendet werden. [12]

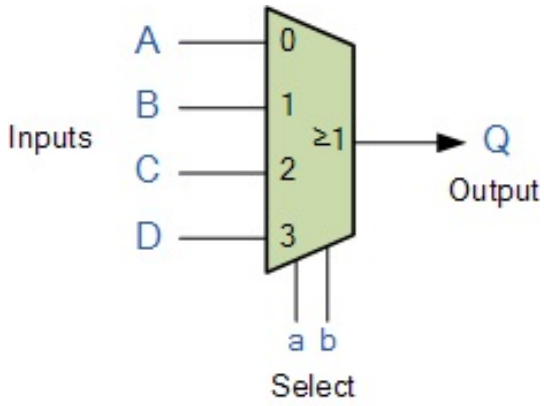

Abbildung 2: Schema eines Multiplexers. [2]

2.1.3 Multiplexer

Ein Multiplexer ist eine Schaltung, mit der aus mehreren Signalen ein Ausgangssignal gewählt werden kann. In einem FPGA kann der Multiplexer dazu verwendet werden, statt des Ausgangssignals des LUT den Wert des Flip-Flops ausgeben zu können. [17]

2.1.4 Beispiel für einen CLB

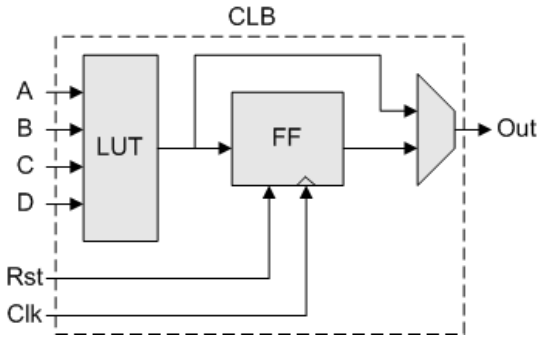

Abbildung 3: Aufbau eines CLB, der aus den Elementen LUT, Flip-Flop und Multiplexer besteht. [3]

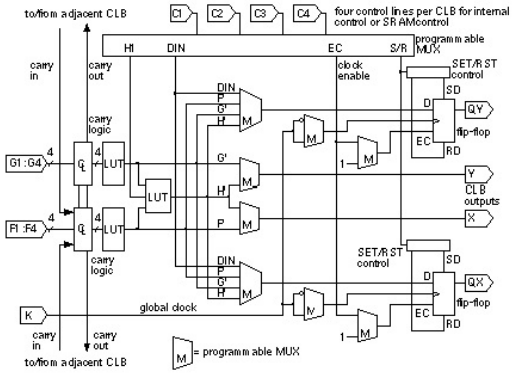

Abbildung 4: Aufbau des XC4000 Logik-Blocks. [4]

Der genaue Aufbau eines CLBs ist je nach Modell unterschiedlich. Dies führt unter anderem dazu, dass die Anzahl der Logikblöcke als Leistungsindikator eines FPGAs nicht sinnvoll ist. So gibt es beispielsweise CLBs wie etwa das ACT 1 Logikmodul, deren Leistung etwa äquivalent zu drei bis vier NAND-Gattern ist während andere deutlich leistungsstärker sind. [7] Auch die Einheit Gates bzw Gate-Equivalents, die häufig in Datenblättern verwendet wird [9], hilft hierbei nicht, da es keine standardisierte Umrechnung von CLBs in NAND-Gatter gibt. In Abbildung 4 ist der Aufbau des Logikblocks XC4000 zu sehen, der so in den Serien XC4000E bzw. XC4000Z von Xilinx zum Einsatz kommt, und aus dem Jahre 1999 stammt. Im Spitzenmodell, dem Xilinx XC4085XL, kommen 3136 solcher Logikblocks zum Einsatz. [19] Aus der vorherigen Erklärung erkennbar sind die Elemente LUT (Look-Up-Tabelle), M (Multiplexer) sowie am rechten Rand die Flip-Flops.

2.2 Zusammensetzung der FPGAs

Wie bereits in 1 genannt wurde, bestehen FPGAs aus hunderten bzw. tausenden dieser logischen Blöcke. In Modernen Chips werden mehrere dieser einfachen CLBs als *Slices* zusammengelegt. Dabei können wie z. B. bei der Virtex-6-Serie einige dieser Slices fest für logische Schaltungen vorgesehen sein, während andere auch als RAM verwendet werden können. Diese sind wie in Abbildung 5 dargestellt miteinander verbunden und verfügen über einen Block-RAM, in dem die Konfiguration gespeichert wird, sowie über I/O, sodass die Daten vom Restsystem bzw. an das Restsystem übergeben werden können.

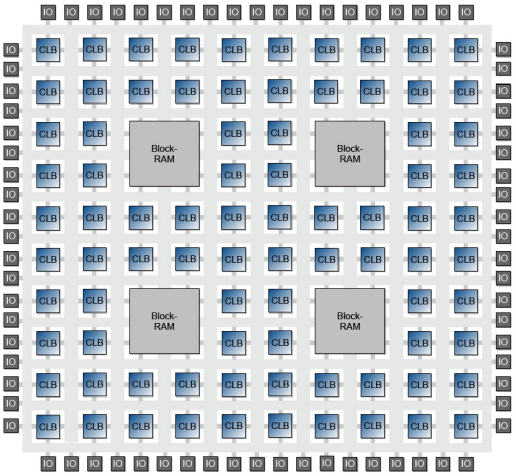

Abbildung 5: Grundlegender Aufbau eines FPGAs bestehend aus mehreren konfigurierbaren Logikblöcken. [15]

3 Programmierung von FPGAs

Traditionell werden FPGAs mit Hardwarebeschreibungssprachen wie z. B. VHDL programmiert, in den letzten Jahren hat sich allerdings über OpenCL eine Möglichkeit aufgetan, FPGAs über imperative Programmierung zu konfigurieren. [11] Intel beispielsweise bietet hierzu sogar ein SDK an. [5] Für die Programmierung mit Hardwarebeschreibungssprachen gibt es z. B. von Xilinx sehr ausführliche Dokumente. [6] Im Folgenden soll anhand eines einfachen Beispiels das Programmieren von FPGAs in VHDL dargestellt werden, bevor ein neuerer Zweig der Forschung angeschnitten wird.

3.1 Programmierung via Hardwarebeschreibungssprachen

Der Schaltungsentwurf eines FPGAs über eine Hardwarebeschreibungssprache besteht aus mehreren Schritten - dem Schaltungsentwurf, einer Simulation, der Synthese, dem Place & Route und der tatsächlichen Konfiguration. Auf diese soll im Folgenden eingegangen werden.

3.1.1 Schaltungsentwurf

Zunächst muss der Aufbau der Schaltung spezifiziert werden. Hierbei soll als Beispiel die Schaltung, die im Kapitel 2 vorgestellt wurde, implementiert werden. Zuerst muss hierbei eine *Entity* dafür geschrieben werden, die Ein- und Ausgabe der Schaltung beinhaltet. Diese Schaltung hat drei Inputs (in der Tabelle 1 als A, B und C bezeichnet) sowie einen Output. Anschließend muss das Verhalten der Schaltung spezifiziert werden. Dies funktioniert über eine *architecture*. Dadurch, dass es sich um eine einfache logische Operation handelt, lässt sich diese Implementierung als Einzeiler lösen. [16]

Listing 1: Implementierung der in Tabelle 1 definierten Schaltung

```
entity ExampleEnt is
        Port (in1  :  in    STD_LOGIC;
         in2       :  in    STD_LOGIC;
         in3       :  in    STD_LOGIC;
         out1      :  out   STD_LOGIC);
  end ExampleEnt;


architecture ExampleArch of ExampleEnt is
        begin
                out1<=(in1 AND in2) or in3;
        end ExampleArch;
```

3.1.2 Simulation

Im Gegensatz zum gewöhnlichen Programmieren lässt sich für FPGAs entwickelter Code nicht so einfach testen, da die Architektur der Zielmaschine von der eines normalen Mikroprozessors völlig abweicht. Ein Testlauf vor dem Überspielen auf das FPGA ist aber trotzdem vorteilhaft, da eine Fehlersuche dann sehr zeitaufwendig sein kann. Um dem Programmierer eine Möglichkeit zu geben, seine Schaltung auf seiner eigenen Maschine zu testen, bringen praktisch alle Entwicklungsumgebungen eigene Software zur Simulation des Schaltungsentwurfs auf sogenannten Testbenches mit. Durch diese kann deutlich zeitsparender die Funktionalität

geprüft werden, da sie verschiedene Werkzeuge mitbringen, über die alle Signale einfach einsehbar sind. Bei Testläufen an der Hardware dagegen tritt oftmals das Problem auf, dass einzelne Signale gar nicht einsehbar sind. Außerdem können fehlerhafte Konfigurationen die Hardware unter Umständen sogar beschädigen. Moderne Entwicklungssysteme erlauben nicht nur die Simulation einzelner Bausteine, sondern auch die Simulation ganzer Systeme - inklusive der dazugehörenden Interaktion der einzelnen Bausteine. [16]

3.1.3 Synthese

Nachdem die Schaltung entworfen und ausgiebig getestet wurde, muss diese in eine Netzliste übersetzt werden werden. Die Netzliste enthält die verwendeten Ressourcen sowie deren Verschaltung, Platzierung und Startzustände spezifiziert. Hierbei können auch Zeitkriterien definiert werden, die beschreiben, wie schnell die schlussendliche Schaltung sein soll. [16]

3.1.4 Place & Route

Zum Schluss erfolgt der Schritt Place & Route - Platzierung und Verdrahtung. Hierbei werden die Komponenten der Netzliste auf reale Hardware abgebildet. Die Ausgabe der Platzierung und Verdrahtung ist ein Bitstrom, der so in das FPGA geladen werden kann. Dieser Vorgang kann nur per Herstellersoftware durchgeführt werden, da die Hersteller i. d. R. den Aufbau des Bitstroms geheim halten. Je nach Schaltungsentwurfes und der gewünschten Optimierung kann dieser Vorgang mehrere Stunden dauern. Dies ist der erste Schritt, dessen Ergebnis von der Zielarchitektur abhängt. Dieser Schritt ist notwendig, da - wie im Kapitel 2 geschildert - keine einheitliche FPGA-Architektur existiert. [16]

3.1.5 Nutzung auf dem FPGA

Im Anschluss wird der generierte Bitstrom auf den FPGA geladen, man sollte allerdings den Bitstrom auch nach dem Laden auf den FPGA nicht löschen, da der Speicher eines FPGA im Regelfall flüchtig ist, das heißt, die Konfiguration ist nach Neustart verloren. An diesem Punkt empfiehlt es sich, die Software trotz der vorherigen Simulation erneut zu testen, da in dieser Phase aus verschiedenen Gründen Fehler auftreten können, die so in der Simulation nicht sichtbar waren: Dies kann daran liegen, dass keine Simulation alle möglichen Abläufe testen kann, aber auch daran, dass die Spezifikation der simulierten Umgebung von der des genutzten FPGA abweicht. [16]

3.2 Weitere Methoden

Da viele Programmierer in Hardwarebeschreibungssprachen nicht so versiert wie in bspw. C++-Programmierung sind, gibt es weitere Methoden um FPGAs zu konfigurieren, zum Beispiel über CHDL [16] oder OpenCL [11]. Dadurch soll erreicht werden, dass Programmierer neben ihnen bekannten Programmiersprachen auch ihnen bekannte Entwicklungsumgebungen nutzen können, weil vor der Platzierung und Verdrahtung sämtliche Aufgaben auch in diesen umgesetzt werden können, wobei eine Netzliste generiert wird, die die Software für Place & Route dann verwerten kann.

4 Anwendung von FPGAs im Hochleistungsrechnen

Durch die, wie in den vorherigen Kapiteln dargestellte, einfache Konfigurierung haben FPGAs im Hochleistungsrechnen immer mehr an Relevanz gewonnen. Während zu Beginn der Technologie die Kosten für FPGAs zu hoch waren, haben sich über die letzten Jahre die Kosten pro Logikzelle deutlich gesenkt - zwischen 1999 und 2009 um Faktor 10 - während sich die Rechenleistung derselben um Faktor 9 erhöht hat (vgl Abbildung 6). Weiterhin ist die massive Parallelisierung, die über FPGAs dank der Umkonfigurierung möglich ist, gerade im Hochleistungsrechnen enorm hilfreich.

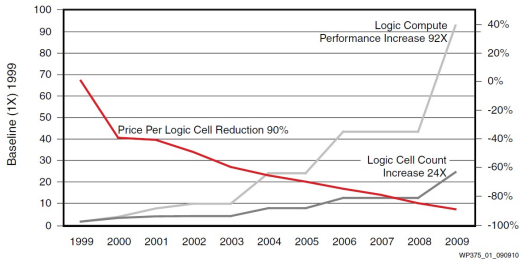

Abbildung 6: Die Preisentwicklung der FPGA. [10]

4.1 Beispiele im Supercomputing

Allerdings hat sich wegen der komplexen Anbindung eines FPGAs an Peripherie ein Konzept etabliert, das sich wohl am besten als Hybrid beschreiben lässt:
Die FHPCA (FPGA High-Performance Computing Alliance) hat bereits 2007 ein System entworfen, bei dem 32 Intel CPUs jeweils zwei Xilinx FPGAs der Virtex-4 Serie steuerten. [14]
Sie testeten in drei Benchmarks die Performance dieser Konfiguration aus: Eine Simulation des Black-Scholes-Modells aus der Wirtschaft, über das Börsenkurse geschätzt werden, einer Bildverarbeitung, bei der aus mehreren Bildern eine dreidimensionale Ansicht erstellt werden soll, sowie einer physikalischen Simulation. Als Ergebnis stellten sie eine Beschleunigung um Faktor 320 beim Black-Scholes-Modell fest, während die deutlich datenintensiveren Aufgaben nur um Faktor 2,5 (Bildverarbeitung) bzw. 5,5 (physikalische Simulation) schneller wurden. Dabei stießen sie auf drei Hürden, nämlich die hohen Entwicklungskosten, die nicht veränderten Limits bei der Speicherbandbreite sowie den Bedarf an parallelisierbarem Code (vgl. Amdahls Gesetz)
Bisher gibt es - den Supercomputer Maxwell ausgenommen - keine großen Supercomputer im wissenschaftlichen Bereich, die auf FPGAs aufbauen. Allerdings nutzt auch Microsofts Cloud seit diesem Jahr einen Hybriden aus normalem Server und FPGAs, um Daten weit schneller verarbeiten zu können.

4.2 Verwendung und Beurteilung von FPGAs im gewöhnlichen HPC

Die Verwendung von FPGAs bietet im Hochleistungsrechnen zahlreiche Chancen, die, nachdem die Leistungsgrenzen herkömmlicher Prozessoren mehr und mehr erreicht scheinen, von den Chipherstellern aggressiver genutzt werden. So erwarb Intel im Dezember 2015 Altera für geschätzte 16,7 Mrd $, einen weiteren FPGA-Hersteller. [13]
Dadurch, dass FPGAs frei konfigurierbar sind, können die einzelnen Logikschaltungen zur massiven Parallelisierung genutzt werden. Dies führt zum einem zu einer erhöhten Rechenleistung, wie schon beim Supercomputer Maxwell festgestellt, aber auch wegen der geringeren Taktfrequenzen zu einem deutlich verringerten Stromverbrauch, moderne FPGAs arbeiten schließlich mit Taktfrequenzen im Bereich von 100-300 MHz. Altera vergleicht in einem Whitepaper die Leistung von herkömmlicher CPU mit der eines FPGA und nennt dabei z. T. die benötigten Taktfrequenzen. [8] Dabei fällt auf, dass praktisch alle Algorithmen trotz der geringeren Taktfrequenz signfikant, zum Teil sogar 100 mal schneller sind. Xilinx stellt einen ähnlichen Vergleich an [1] und kommt auch zu ähnlich deutlichen Ergebnissen.

[1] [10], Seite 7

4.2.1 Koprozessoren

Auch bietet das Einsatzgebiet der FPGA-Koprozessoren massive Chancen. Schon heute werden diese in Bereichen wie Bildverarbeitung, Verschlüsselung oder Datenkompression eingesetzt - Gebiete, an denen normale Mikroprozessoren nicht nur aufgrund ihrer Leistung, sondern vor allem wegen ihrer Anbindung an den Hauptspeicher an ihre Grenzen stoßen. Die Idee dabei ist, dass der gewöhnliche Mikroprozessor diejenigen Aufgaben, die durch einen FPGA kaum schneller laufen, ausführt, und den Rest vom speziell dafür konfigurierten FPGA lösen zu lassen.
Ein solches System besteht aus:

- FPGA(s)
- den lokalen Speicher des FPGA
- ein konfigurierbares Taktsystem
- eine Schnittstelle zur Konfiguration des/der FPGA, um diese auf den jeweiligen Algorithmus anzupassen
- I/O
- Ein Interface für Schreib- und Lesezugriffe auf den Speicher des Mikroprozessors

Durch Koprozessoren können einige Einschänkungen, die durch die von-Neumann-Architektur bestehen, umgangen werden. Die Verwendung von FPGAs hat sich hierbei gegen anderen Konzepte, wie etwa einen zusätzlichen Chip auf dem Prozessor, durchgesetzt. Durch die Rekonfigurierbarkeit ist sichergestellt, dass, wenn genug Logikeinheiten frei sind, diese immer die aktuellen Befehle ausführen können - etwas, das bei ASICs (Application Specific Integradted Circuit) keineswegs der Fall sein muss. [16] [10]

4.2.2 Einsatz als Schnittstelle

Ein weiteres verbreitetes Anwendungsgebiet von FPGAs ist als Schnittstelle zwischen unterschiedlichen Systemen - bspw. als Schnittstelle zwischen

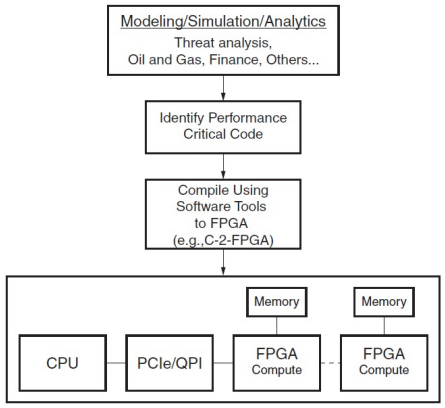

Abbildung 7: Aufbau eines FPGA-Koprozessor. [10]

Sensoren und CPUs. Dadurch, dass sämtliche, sonst in Software gelösten Konfigurationen, hardwareseitig implementiert werden können, werden so deutlich höhere Durchsätze erreicht. [10]

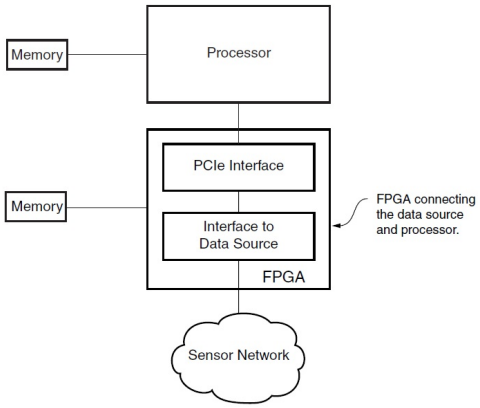

Abbildung 8: Nutzung eines FPGA als Interface zwischen Mikroprozessor und Sensor. [10]

4.2.3 Einsatz als Hardwarebeschleuniger bei konstanten Funktionen

Aufgrund ihres hohen Durchsatzes, der Fähigkeit viele Daten auf einmal zu verarbeiten und der geringen Latenz werden FPGAs häufig genutzt, um

große Datenmengen mit der gleichen Funktion zu bearbeiten. Ein Beispielaufbau findet sich in 9.

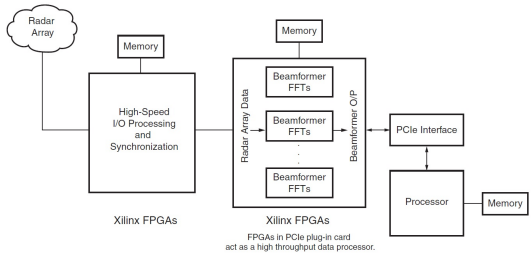

Abbildung 9: Nutzung eines FPGA als Hardwarebeschleuniger bei konstanten Funktionen. [10]

4.2.4 Beurteilung

Neben den in den vorherigen Beispielen angesprochenen Vorteilen können Konzepte wie Pipelining noch besser umgesetzt werden, da ausgeschlossen ist, dass auf bestimmte funktionelle Einheiten gewartet werden muss. Eine Stärke, die insbesondere an diesem Punkt sichtbar wird, ist die Tatsache, dass FPGAs zwar flexibel konfigurierbar sind, aber am Ende trotzdem eine hardwarebasierte Lösung und damit sehr performant sind.
Im Falle von sequentiellen Algorithmen zeigt sich allerdings die geringe Taktfrequenzen der FPGAs, da diese dort deutlich weniger performant als gewöhnliche Prozessoren sind - ein Problem, das durch die Koprozessortechnik gelöst wird. Gerade die geringen Taktfrequenzen werden bei sequentiellen Programmen auf Dauer eine große Schwierigkeit darstellen, wenn auf zusätzliche Mikroprozessoren verzichtet werden soll, da fest verdrahtete Schaltkreise aus physikalischen Gründen mit deutlich höheren Frequenzen arbeiten können. Weiterhin ist der Stromverbrauch eines FPGAs zwar im Mittel besser als der eines Mikroprozessoren, aber in einzelnen Operationen reicht er nicht an den eines darauf optimierten ASIC heran. Darüber hinaus sind FPGAs bei Gleitkommaoperationen noch nicht unbedingt sinnvoller als Graphikkarten, deren explizit darauf spezialisierte ASICs hier natürlich deutlich höhere Geschwindigkeiten erreichen können.
Ein weiterer Nachteil SRAM-basierter FPGAs ist es, dass deren Konfiguration bei Neustart erneut geladen werden muss, wozu ein zusätzlicher nichtflüchtiger Speicher nötig ist.

5 Ausblick und Fazit

5.1 Ausblick auf zukünftige Entwicklungen

In den vergangenen Kapiteln wurde gezeigt, wie FPGAs schon heute einen maßgeblichen Beitrag im HPC leisten können, doch es ist vor allem die Zukunftsperspektive, die sie zu einem derart interessanten Thema macht. Schon heute besteht ein Markt für performantere Hardware. Wie aus Abbildung10 hervorgeht, geht man heutzutage davon aus, dass der moderne Rechenbedarf diejenige Leistungsfähigkeit, die Prozessoren nach dem Moore'schen Gesetz haben können, übersteigt - insbesondere bei der Verarbeitung von großen Datenmengen. Unternehmen wie Intel scheinen - der Zukauf von Altera erweckt zumindest diesen Eindruck - das Potenzial davon erkannt zu haben, FPGAs in eine promintere Stellung zu rücken, da die Technologie auch genutzt werden muss, um helfen zu können. Eine große Stärke der FPGAs - das hervorragende Skalieren der Speicherbandbreite (vgl. Abbildung 11) - ist gerade ein gravierendes Problem der heutigen Mikroprozessoren.
Ein weiteres Problem - nämlich die hohen Kosten - wird durch FPGAs ebenfalls reduziert, da diese im Gegensatz zu ASICs für eine Vielzahl an Problemen genutzt werden können. Weiterhin liegt der Stromverbrauch eines FPGAs z. T. unter der einer deutlich langsameren GPU bzw. GPP [2]

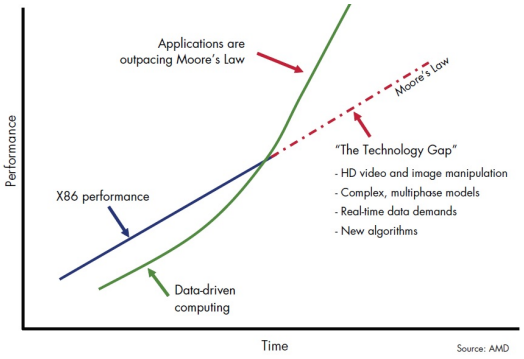

Abbildung 10: Diskrepanz zwischen Leistungsfähigkeit herkömmlicher Prozessoren und den gestiegenen Anforderungen. [8]

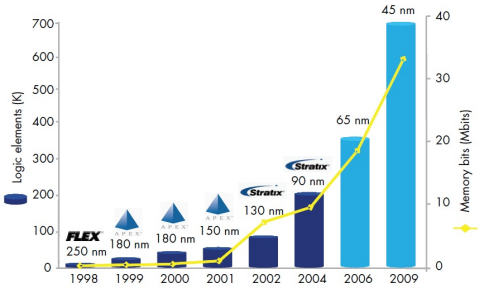

Abbildung 11: Entwicklung der Speicherbandbreite von FPGAs. [8]

5.2 Fazit

Auf den letzten Seiten wurde eine Technologie vorgestellt, die verspricht, eines der Kernprobleme der heutigen Informatik lösen zu können: Das Verarbeiten großer Datenmengen. Es wurde an einigen Beispielen gezeigt, wo dies schon heute funktioniert - z. T. mit erheblichem Mehrwert - und es wurde auch vorgestellt, dass große Unternehmen wie Intel und Microsoft mehr und mehr die Möglichkeiten der Verwendung von FPGAs im Hochleistungsrechnen als Anreiz sehen, in diesen Bereich zu investieren. Außerdem wurden klare Schwachpunkte der Technik dargelegt - inbesondere die Abarbeitung sequentieller Algorithmen - aber auch die Relevanz dieser Technologie für die Zukunft. Vermutlich stellen die in 4.2 dargestellten Bereiche die zentralen Anwendungsgebiete über die nächsten Jahre dar, da das eben genau die Stellen sind, an denen herkömmliche Prozessoren nicht mehr mithalten können. Durch den Zukauf Alteras durch Intel kann man vermuten, dass es in Zukunft für Entwickler einen deutlich einfachereren Zugang zur Entwicklung auf FPGAs gibt als die für die meisten Entwickler doch recht ungewohnten Hardwarebeschreibungssprachen, wodurch sich die Verbreitung dieses Wissens deutlich verbreiten sollte.

[2] vgl. [18], Seite 76

Literatur

[1] https://www.express-elect.com/part_image/XC2064-PC68-70C2470.jpg.

[2] http://www.electronics-tutorials.ws/combination/comb26.gif?x98918.

[3] https://upload.wikimedia.org/wikibooks/en/c/ca/CLB_Block_Diagram.png.

[4] http://www10.edacafe.com/book/ASIC/CH05/CH05-7.gif.

[5] https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[6] Constraints guide. https://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf.

[7] ACT 1 Series FPGAs, 1996. http://www.microsemi.com/document-portal/doc_view/130666-act-1-series-fpgas.

[8] Accelerating High-Performance Computing With FPGAs. White Paper, 2007. https://www.altera.com/en_US/pdfs/literature/wp/wp-01029.pdf/ Version 1.1.

[9] Spartan-II FPGA Family Data Sheet, Juni 2008. https://www.xilinx.com/support/documentation/data_sheets/ds001.pdf.

[10] High Performance Computing Using FPGAs. White Paper, 2010. http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Using_FPGAs.pdf.

[11] Implementing FPGA Design with the OpenCL Standard. White Paper, 2013. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01173-opencl.pdf/ Version 3.0.

[12] Wie funktionieren FPGAs? White Paper, 2013. http://www.ni.com/white-paper/6983/de/.

[13] Intel Completes Acquisition of Altera, 2015. https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/.

[14] Dr. Rob Baxter. Maxwell: a 64-FPGA Supercomputer, 2007. http://rssi.ncsa.illinois.edu/docs/academic/Baxter_presentation.pdf.

[15] Robert Dietrich. SGI RASC: Evaluierung einer Programmierplattform zum Einsatz von FPGAs als Hardware-Beschleuniger im Hochleistungsrechnen. Diplomarbeit, Technische Universität Dresden, 2009. http://queens.inf.tu-dresden.de/dietrich_diplom.pdf.

[16] Klaus Kornmesser. *Das FPGA-Entwicklungssystem CHDL*. PhD dissertation, Universität Mannheim, 2004. https://ub-madoc.bib.uni-mannheim.de/857/1/kornmesser.pdf.

[17] Markus Wannemacher. *Das FPGA-Kochbuch*. 1998.

[18] Wim Vanderbauwhede and Khaled Benkrid. *High-Performance Computing Using FPGAs*. 2013.

[19] Xilinx. *XC4000E and XC4000X Series Field Programmable Gate Arrays*, 1999. https://www.xilinx.com/support/documentation/data_sheets/4000.pdf, Version 1.6.