UNIVERSITÄT LEIPZIG

Forschungsseminar Parallele Algorithmen und Komplexe Systeme

Ausarbeitung zum Paper *„Cluster of re-configurable nodes for scanning large genomic banks“* (DA1)

Prof. Middendorf, Dr. Merkle
Arne Brutschy[1], 13. Dezember 2005

Zusammenfassung

Dieser Artikel beschäftigt sich dem Paper *„Cluster of re-configurable nodes for scanning large genomic banks“* von D. Lavenier et. al.

Aufgrund ihrer stetig wachsenden Größe sind Genom-Datenbanken mit konventionellen Methoden kaum noch erfolgreich zu bewältigen. Die Autoren stellen in ihrer Arbeit ein Projekt vor, das die Performanz von häufigen Anwendungen in der Bioinformatik wie Muster- und Ähnlichkeitssuche auf solchen Datenbanken durch Parallelisierung und Hardware-Implementierung der verwendeten Filter erhöht. Die Umsetzung dieser Vorschläge durch einen Cluster von FPGAs wird in diesem Artikel erläutert und anschließend diskutiert.

Einleitung

Eines der Hauptarbeitsfelder in der Molekularbiologie ist die Erforschung der Funktion von Genen. Biologen klassifizieren dabei die Homologie zweier Gene anhand übereinstimmender oder ähnlicher Sequenzen von Aminosäuren. Dies ermöglicht es, die Evolution von Lebewesen nachzuvollziehen und von bekannten auf unbekannte, aber wahrscheinlich in der Funktion ähnliche Gensequenzen zu schließen.

Aus diesem Grund werden täglich große Datenbanken mit Geninformationen von Biologen nach Mustern durchsucht. Dabei gibt es aber das Problem, dass die Menge der Daten exponentiell wächst – genauer gesagt verdoppelt sie sich ungefähr alle zwölf Monate. Da sich aber nach Moore's Law die Leistungsfähigkeit der Prozessoren nur ungefähr alle achtzehn Monate verdoppelt [1], müssen hier alternative Lösungswege gefunden werden, um mit der Entwicklung der Gendaten Schritt zu halten.

In dem Paper *„Cluster of re-configurable nodes for scanning large genomic banks“* schlagen die Autoren eine Lösung zur performanten Verarbeitung solcher Suchen auf Gendaten vor [2]. Dabei machen sie sich zu Nutze, dass ein Großteil der Daten in einem ersten Schritt der Algorithmen als uninteressant klassifiziert werden kann und somit nicht in einer aufwändigen Operation weiter untersucht werden muss. Durch Implementierung dieser Teile des Algorithmus in rekonfigurierbarer Hardware und Verteilung der Datenbank auf mehrere Nodes eines Clusters wird versucht, eine hohe Performanz zu erzielen. Die im Paper präsentierte Lösung wird in diesem Artikel genauer erläutert.

Zuerst werden der Hintergrund und grundlegende Algorithmen zur Sequenz-

[1]MatrNr: 8964813, abrutschy@xylon.de

und Mustersuche, dann die vorgeschlagene Lösung im Detail und ihre Implementierung in rekonfigurierbarer Hardware vorgestellt. Anschließend werden die Testresultate besprochen und das gesamte Projekt abschließend diskutiert.

Hintergrund

Um zu einer gegebenen Abfragesequenz ähnliche, in ihrer Funktion oder Abstammung vielleicht übereinstimmende Sequenzen aus einer gegebenen Menge von Gendaten zu extrahieren, wird üblicherweise ein so genanntes Sequenzalignment durchgeführt. Dabei werden die Unterschiede zwischen der Abfragesequenz und den Daten wie zum Beispiel Gaps, Insertions oder Deletions nach einem Punktesystem bewertet. Bei der Suche werden dann nur Alignments als *Hit* bewertet, wenn ihre Bewertung einen gegebenen Schwellwert überschreitet. Auf diese Weise werden uninteressante oder wenig ähnliche Sequenzen effektiv ausgefiltert.

Der zu diesem Zweck 1981 von Smith und Waterman vorgeschlagene Algorithmus ist allerdings sehr kostenintensiv und wird heutzutage eher selten in seiner Urform verwendet [3]. Indes bauen die meisten modernen Verfahren zum Sequenzalignment auf diesen Algorithmus auf.

BLAST

BLAST (Basic Local Alignment Search Tool) ist heutzutage die wohl am häufigsten in der Bioinformatik gebrauchte Anwendung [4]. Dabei approximiert BLAST den Algorithmus von Smith-Waterman, mit dem Unterschied das die Entwicklungsziele auf hoher Geschwindigkeit bei relativ guten Erkennungsraten liegen. Die Gewichtung auf Geschwindigkeit ist bei den riesigen Gendatenbanken heutzutage notwendig, um ein effizientes Sequenzalignment überhaupt erst zu ermöglichen.

BLAST erreicht durch die Aufteilung des Sequenzalignments in drei Schritte die bis zu 50-fachen Geschwindigkeit gegenüber Smith-Waterman. Die Idee des Algorithmus basiert auf der Wahrscheinlichkeit, dass Alignments kurze Stücke von hoher Identität besitzen. Diese Teilstücke werden dann während der Suche nach besseren und längeren Alignments weiter vergrößert. Indem diese Segmente kurzgehalten werden, ist es möglich, die Abfragesequenz vor einer Suche zu bearbeiten und eine Tabelle aller möglichen Teilstücke mit ihrem Ursprung in der Originalsequenz vorzuhalten. Dabei stellt der Algorithmus eine Liste aller benachbarten Worte fester Länge auf, die einen Treffer auf der Abfragesequenz mit einem höheren Score als ein zu wählender Parameter erzeugen würden. Anschließend wird die Zieldatenbank nach Worten in dieser Liste abgefragt und die gefundenen Treffer erweitert, um mögliche maximale zusammenhängende Treffer in beiden Richtungen zu finden [5]. Die Schritte sind dabei im Einzelnen:

1. Begonnen wird mit einer Suche nach kurzen Übereinstimmungen mit einer fixen Länge w zwischen der Suchsequenz und den Sequenzen in der Datenbank.
2. Die gefundenen Sequenzen werden in beide Richtungen erweitert um das Scoring der jeweiligen Sequenz zu erhöhen. Dabei werden nur übereinstimmende Abschnitte bewertet, Lücken oder Mutationen werden in diesem Schritt nicht berücksichtigt.
3. Falls im zweiten Schritt eine Sequenz gefunden wurde, die einen gegebenen Schwellwert überschreitet, wird für sie ein vollständiges Alignment ähnlich dem von Smith-Waterman durchgeführt und das Resultat an den Nutzer gemeldet.

Heutzutage sind viele Variationen von BLAST in Benutzung, so zum Beispiel Psi-BLAST, BLAT sowie parallele Implementierungen derselben [6, 7].

Pattern Search

Eine andere Klasse von Suchen stellt die Mustersuche (*pattern search*) dar, die zum Suchen von Proteindomänen

gebraucht wird. Proteindomänen sind Gruppen von Proteinen, mit deren Hilfe man die entsprechenden Gene Gruppen und Familien zuzuordnen kann und die dadurch Rückschlüsse auf ihre Funktion erlauben. Im Gegensatz zum einfachen Sequenzalignment werden hier bekannte Muster von Gruppen von Aminosäuren (Colons) in der Datenbank gesucht. Diese stammen normalerweise aus Datenbanken wie der PROSITE pattern database [8]. Diese öffentlich zugänglichen Beschreibungen von Mustern listen Funktion, Verwandtschaften und ähnliches von passenden Gensequenzen auf.

Die Muster werden üblicherweise mittels regulären Ausdrücken dargestellt und können somit durch endliche Automaten implementiert werden. Ein Beispiel ist das Muster `D - [ILV] - x(1,3) - A`, welches sich aus den Aminosäuren `D` am Anfang, gefolgt von `I`, `L` oder `V`, einer Lücke von ein bis drei beliebigen und `A` am Ende zusammensetzt.

Die Mustersuche wird von verschiedenen Programmen implementiert, so zum Beispiel Patternhunter [9].

Bisherige Lösungen

Die genannten Programme sind die verbreiteste Grundlage für heutige Lösungen in diesem Bereich. Aufgrund der guten Parallelisierbarkeit von Algorithmen wie BLAST werden diese häufig auf Multiprozessormaschinen oder Clustern von Computern, vornehmlich günstige PCs „von der Stange“, eingesetzt.

Eine Gendatenbank wird dabei gleichmäßig über die Nodes eines Cluster verteilt, so dass die Algorithmen lokal auf den Daten arbeiten können. Da die einzelnen Tasks voneinander Unabhängig sind, ist nur eine sehr geringe Menge an inter-node Kommunikation notwendig. Dies schaltet den häufigen Performanzengpass bei der Datenverarbeitung durch Cluster, die limitierte Netzwerkbandbreite, aus.

Kommerzielle Produkte in diesem Bereich beinhalten:

- DeCypher solutions, von Timelogic
- GeneMatcher2, von Celera Genomics
- BioXL/H, von Biocceleration

und andere, die allesamt entweder sehr spezielle Probleme bearbeiten oder nur eine weitere Form eines einfachen Clusters darstellen[2].

Verwandte Arbeiten

Die Autoren schlagen in ihrem Paper eine FPGA-basierte Lösung vor, die es ermöglicht die Sequenzen der Datenbank direkt am Ausgang der Datenspeicher zu filtern und so nur relevante Abschnitte an den verarbeitenden Computer weiterzureichen. Diese Idee ist schon älter und wurde unter anderem bereits für diverse Datenbanksysteme vorgeschlagen. Aufgrund der mangelnden Flexibilität dieser Systeme werden sie heute aber nicht mehr eingesetzt.

Derzeit wird in verschiedenen Projekten an verwandten Lösungen gearbeitet, bei denen ebenfalls versucht wird, einen Teil der Berechnungen von den Controllern der Festplatten erledigen zu lassen. Dies ermöglicht eine einfache Parallelisierung und eignet sich sehr gut zum on-the-fly Filtern großer Datenbestände. Zusätzlich skalieren solche Lösungen im Allgemeinen sehr gut. Daher setzen aktuelle Projekte wie

- IDisk
- Active Disk
- Smart Disk

diese Techniken für Aufgaben wie Datamining, Verschlüsselung und Nachrichtendienstliche Zwecke ein[2].

Hardwarerealisierung

Um Gensequenzen effektiv am Datenspeicher filtern zu können, schalten die Autoren FPGA-Boards direkt vor handelsübliche Festplatten. Die rekonfigurierbare Logik übernimmt dann Zugriff auf die Daten, Filtern derselben und

[2] Für Referenzen zu den einzelnen Produkten bzw. Projekten siehe [2].

Meldung der Ergebnisse über das Netzwerk. Im vorgestellten Projekt wurden 48 solcher Nodes in einem Cluster zusammengeschaltet. Ein „Hostcomputer“ übernimmt dabei die Steuerungsfunktion und erledigt das Postprocessing der relevanten Daten. Die einzelnen Nodes des Clusters werden über einfaches 100 Mbit/s Ethernet zusammengeschaltet.

Aufbau einer RDISK

Ein RDISK-Knoten setzt sich aus der Festplatte und einem low-end FPGA-Prozessor zusammen, die gemeinsam auf einer Platine im PCB-Format montiert werden [10]. Dies erlaubt es, die einzelnen Knoten einfach in einem Rack zu montieren; bis zu acht Knoten können dabei auf einer Ebene angebracht werden. Das Ziel des Projektes ist ein performantes Sequenzalignment bieten zu können, das im Vergleich zu sonst üblichen PC-Clustern kosteneffektiv arbeitet. Durch Wahl von Hardwarekomponenten, die in großen Stückzahlen hergestellt werden, werden geringe Kosten des Gesamtsystems garantiert. Das System setzt sich aus den folgenden Komponenten zusammen:

- Xilinks Spartan-II FPGA
- MC-Unit
- Ethernet Controller
- 32 MB Ram
- Festplatte
- PCB-Platine

Durch die konsequente Verwendung günstiger Hardware bleiben die Gesamtkosten für ein solchen RDISK-Knoten unter 200 $. Der Aufbau eines RDISK-Node wird in Abbildung 1 gezeigt.

System on a Chip

Auf dem FPGA-Prozessor wurde ein komplettes System on a Chip (SoC) implementiert. Dabei wurde auf ein bestehendes System zurückgegriffen, welches möglichst wenig Platz auf dem FPGA einnimmt. Das System übernimmt dabei die komplette Ansteuerung der Festplatte, des Netzwerkchips inklusive Protokolldaten, Kontrolle und Überwachung der Hardwarefilter und leitet Rekonfigurationsanweisungen an die entsprechende Einheit weiter. Das komplette System verbraucht nur knapp die Hälfte der verfügbaren Ressourcen des FPGAs, was ungefähr 3000 LUTs und 5 kB Speicher zur Implementierung von Filtern übrig lässt. Abbildung 2 zeigt ein Strukturdiagramm des Systems.

Das System wird über eine „bootstrap configuration“, die auf einem geschützten Abschnitt der Festplatte liegt, initialisiert. Diese erlaubt zwei Basisoperationen auf dem Node: Dateien (Datenbanken oder Konfigurationen in Form von binärem Code) können auf die Festplatte geladen werden und der FPGA kann mit einer der genannten Konfigurationen rekonfiguriert werden. Die Konfigurationen werden, soweit nicht schon vorhanden, über das Netzwerk nachgeladen und

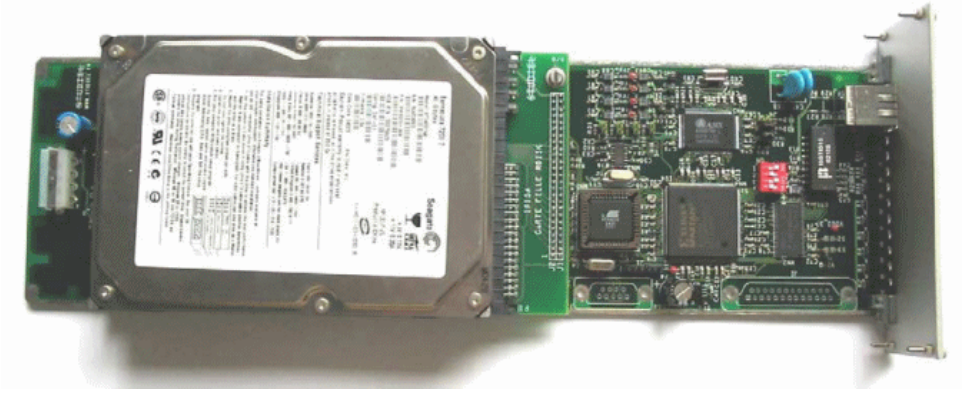

Abbildung 1: Aufbau eines RDISK Nodes

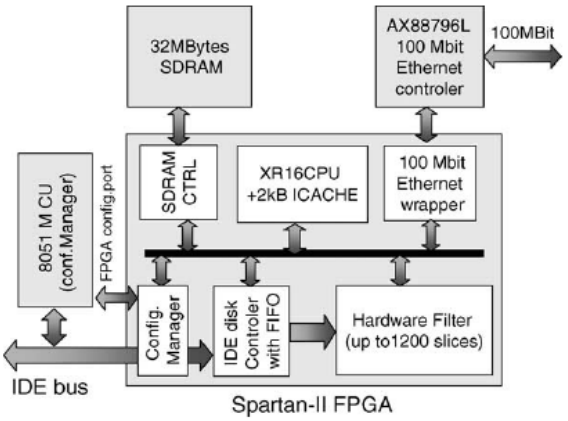

Abbildung 2: Aufbau des System on a Chip

auf der Festplatte zwischengespeichert; sie fungiert demnach als Konfigurations-Cache. Zusätzlich dient die „bootstrap configuration“ für Notfälle, um auch bei Fehlern oder Timeouts auf das System zugreifen zu können.

Sobald der Node mit den entsprechenden Daten versorgt wurde, kann durch Angabe der Datenbank, Konfiguration und Suchausdruck das Filtern der Daten gestartet werden. Dabei werden die Daten von der betreffenden Datei auf der Festplatte gelesen, mittels der implementierten Hardwarefilter beurteilt und relevante Abschnitte mit Offsetinformationen an den Hostcomputer gesandt.

Um möglichst hohen Durchsatz für sequentielle Zugriffe auf die Datendank bei einem gleichzeitig kleinem Basissystem zu gewährleisten, ist das Dateisystem auf der Festplatte ist relativ einfach gehalten. Bei dem verwendeten Netzwerkprotokoll, ein proprietärer Aufsatz auf UDP, verhält es sich ähnlich. Die Erstellung neuer Applikationen erfordert dabei zwei Schritte:

1. Implementierung der Hardware-Filter in VHDL
2. Implementierung des Postprocessing in C

Die Filter-Spezifikation wird mit dem vorgefertigten SoC zusammengeführt und das C-Modul für das Postprocessing mit den entsprechenden Bibliotheken kompiliert. Tools, Templates und andere Hilfsmittel werden zur Verfügung gestellt. Die Autoren veranschlagen ungefähr einen Mannmonat für die Erstellung einer neuen Applikation.

Implementierung der Algorithmen

Die Autoren implementierten und testeten zwei Applikationen im Cluster. Es handelte sich dabei um die oben bereits erklärte Ähnlichkeits- und Mustersuche (*similarity* bzw. *pattern search*).

Ähnlichkeitssuche

Bei der Ähnlichkeitssuche handelt es sich um einen mit der BLAST-Software verwandten Algorithmus. Die grundlegende Idee dieser Applikation ist es, die ersten zwei Schritte des heuristischen Sequenzalignments, der Suche von möglichen Hits, vom FPGA direkt am Ausgang der Datenspeicher durchführen zu lassen. Wie bereits gesagt werden dann nur statistisch relevante Daten an den Hostcomputer weitergereicht. Das endgültige Berechnen des Alignments nach Smith-Waterman geschieht dann in der Postprocessing-Phase und benötigt nicht sehr viel Rechenleistung.

Die Autoren implementieren die Hardware-Filter mit sogenannten „processing elements“ (PEs). Jeder Node enthält mehrere PEs, wobei die Anzahl dieser idealerweise gleich der Anzahl

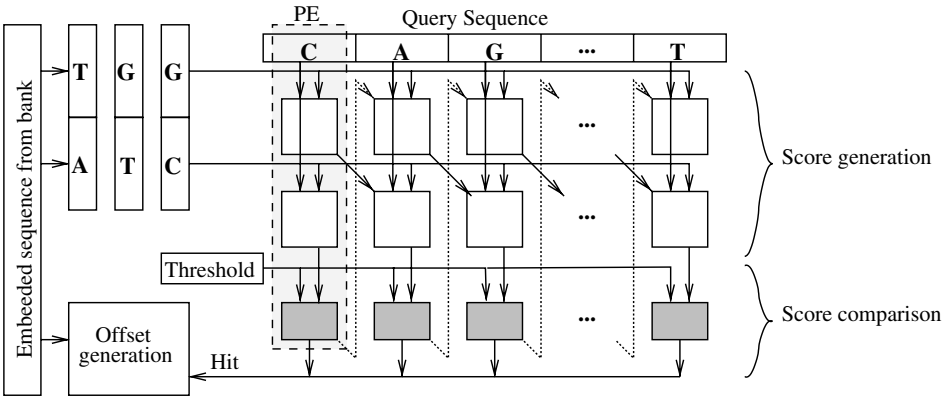

Abbildung 3: Struktur der Filterimplementierung für die Ähnlichkeitssuche. Eine Abfragesequenz `CAG···T` wird hier mit dem Abschnitt `GCGTTA···` aus der Datenbank verglichen.

der Länge der Suchsequenz ist. Dies ergibt sich aus der Funktion der PEs: zu Beginn der Suche nimmt jede Einheit ein Nukleotid der Suchsequenz auf und vergleicht sie mit dem momentan geladenen Abschnitt der Datenbank. Der *Score* wird dabei parallel berechnet und mit dem fest einkodierten Schwellwert verglichen. Wird dieser überschritten, wird der entsprechende Abschnitt mit Offsetinformationen an den Hostcomputer weitergereicht. Je feiner der Filter ist, desto größer ist sein Schwellwert. Die Implementierung desselben wird damit aufwändiger und verbraucht mehr Platz, dadurch sinkt die Zahl der PEs die in den Node integriert werden können - und damit auch implizit die Länge der möglichen Suchsequenzen. Abbildung 3 zeigt den Aufbau der Filterimplementierung und Tabelle 1 listet das Verhältnis von Schwellwert zur Länge der Suchsequenz auf.

Mustersuche

Die Mustersuche kann, da diese regulären Ausdrücken entspricht, durch gewichtete endliche Automaten (WFA) implementiert werden. Ein Automat für das oben bereits erwähnte Beispielmuster `D - [ILV] - x(1,3) - A` ist in Abbildung 4 gezeigt. Die Implementierung solcher Automaten wurde von den Autoren schon in einem gesonderten Paper untersucht [11]. Der WFA wird durch das dort vorgeschlagene *linear encoding scheme* direkt auf die Hardware abgebildet. Dabei entspricht jeder Zustand einem dedizierten Register und jede Transition einem Komparator und Addierer für die Bewertungsfunktion. Dies erhöht für jede erkannte Aminosäure im Automat den Score der aktuellen Sequenz. Wie bei der Ähnlichkeitssuche kann die Operation nur bis zu einem festgelegten Schwellwert durchgeführt werden, die Finalisierung der Suche erfolgt dann im Postprocessing. Alternativ kann die komplette Suche im FPGA durchgeführt werden. Dies ist die in dem Paper diskutierte Variante.

Je nach Größe des implementieren Automaten (die wiederum von der Komplexität des zu suchenden Musters abhängt) können bis zu sechs Automaten implementiert werden. Jeder Automat liest

Schwellwert	programmierbar	16	18	20	22	30
max. Sequenzlänge	137	198	180	160	159	157

Tabelle 1: Verhältnis von Schwellwert zur maximalen Länge der möglichen Suchsequenz (und damit Anzahl von PEs).

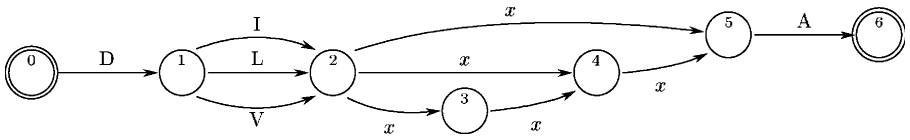

Abbildung 4: Ein gewichteter endlicher Automat (WFA) für das Beispielmuster `D - [ILV] - x(1,3) - A`.

immer nur ein Basenpaar pro Taktzyklus ein. Sechs Automaten erlauben es also jeweils eine komplette Aminosäure pro Taktzyklus in Vorder- und Rückrichtung zu lesen. Hier können also wie bei der Ähnlichkeitssuche bei geringerer Parallelität komplexere Suchmuster implementiert werden. Tabelle 2 listet das Verhältnis von Größe, Anzahl der Automaten und Musterlänge auf.

Tests und Resultate

Die Implementierungen der Algorithmen wurden von den Autoren unter realen Bedingungen getestet. Der Unterschied der Applikationen liegt bei der Rekonfiguration der Nodes: bei der Ähnlichkeitssuche werden vordefinierte Filter mit der Suchsequenz einmalig an die Nodes gesandt. Der Cluster muss nur rekonfiguriert werden, wenn die Art der Suche, nicht aber die Suchsequenz geändert wird. Bei der Mustersuche ist dies anders: für jedes Muster wird eine eigene Konfiguration erzeugt. Der Cluster wird also bei jeder Änderung des Suchmusters zur Rekonfiguration gezwungen.

Ähnlichkeitssuche

Bei einer Datenrate von 15,5 MB/s seitens der Festplatten werden Daten mit einer Geschwindigkeit von 60,4 Mbp/s an die Filter geliefert. Die verwendeten Filter wurden so implementiert, dass Gendaten bei voller Festplattenbandbreite ausgelesen werden können; die Eingabebandbreite liegt mit 78 Mbp/s sogar leicht höher als nötig.

Getestet wurden die Filter an der Primatenabteilung der Genbank, die 4,3 Gbp umfasst. Die verwendeten Suchsequenzen wurden so gewählt, dass sie repräsentative Ergebnisse abdecken; die durchschnittliche Länge lag bei 300 Basenpaaren.

Im Test zeigte sich, dass die Filter von sehr hoher Qualität sind, fast jeder Hit führte zu einem erfolgreichen Alignment. Dies ist nicht weiter verwunderlich, wird im FPGA doch fast eine exakte Prüfung des Alignments durchgeführt. Im Vergleich zu BLAST scheinen die Resultate aber von geringerer Genauigkeit zu sein, die Suche ist nicht so fehlertolerant und „übersieht“ daher einige Treffer. Quantitative Aussagen zu diesem Problem fehlen aber im untersuchten Paper.

Im Performanzvergleich schneidet der RDISK-Cluster hervorragend ab: eine 16-Prozessor SUN Fire 6800 Maschine[3] war im Test ungefähr neunmal langsamer. Ein einzelner RDISK-Knoten ist al-

[3] Diese wurde gewählt, da es sich dabei um die gleiche Prozessorgeneration wie der Spartan-II handelt. Die Ergebnisse sind somit vergleichbar.

Automat		Bandbreite		max. Suchsequenz			
Einheiten	LUTs	pro Cycle	bei 40 Mhz	p=6	p=8	p=10	p=12
x 1	7p	1 bp	40 Mbp/s	71	53	42	35
x 2	14p	1 bp	40 Mbp/s	35	26	21	17
x 3	15p	3 bp	120 Mbp/s	33	25	20	16
x 6	30p	3 bp	120 Mbp/s	16	12	10	8

Tabelle 2: Verhältnis von Größe und Komplexität der Automaten, parallel implementierte Einheiten und maximale Länge des möglichen Suchmusters.

so ungefähr dreimal so schnell wie ein Knoten der SUN.

Mustersuche

Je nach Komplexität des Suchmusters steigt bei der Mustersuche die Größe des zu implementierenden Automaten. Je kleiner dieser ist, desto mehr parallele Einheiten können im Node verwirklicht werden. Bei einer oder zwei Einheiten liegt die Verarbeitungsgeschwindigkeit der Filter bei 40 Mbp/s und bildet so den Performanzengpass des Systems. Bei drei oder sechs Einheiten können bis zu 120 Mbp/s verarbeitet werden, die Nodes operieren also wieder an der maximalen Bandbreite der Festplatten.

Die Tests wurden anhand der EST Canine Database durchgeführt, die 34 Gbp umfasst. Die implementierten Muster hatten zwischen 11 und 25 Transitionen und konnten somit mittels einem zweifachen Automaten realisiert werden.

Im Vergleich zu einem aktuellen PC[4] ist ein einzelner RDISK-Node viermal so schnell. Nach Aussage der Autoren ist damit der RDISK-48 Cluster äquivalent zu einem PC Cluster mit 192 Nodes. Dabei wurde aber nur die reine Rechenzeit verglichen, die Rekonfigurationszeit der einzelnen Nodes liegt im Bereich mehrerer Minuten und wurde nicht berücksichtigt. Tabelle 3 listet eine Zusammenfassung der Testergebnisse auf.

[4]Standard PC mit 2 Ghz, 768 MB Ram

Diskussion

Die Vorteile des Projektes liegen auf der Hand: es bietet einen hochperformanten Cluster, der durch konsequentes Nutzen von Standard-Hardwarekomponenten mit rund 15.000 $ deutlich günstiger ist als ein PC Cluster mit vergleichbarer Leistung. Weitere Vorteile sind im Einzelnen:

- Komplexe Filter können direkt in die Hardware integriert werden, was eine hohe Selektivität erlaubt.
- Datenbanken können mit der maximalen Festplattenbandbreite durchsucht werden.
- Inter-Node Kommunikation ist sehr gering und kann daher über ein günstiges Ethernet-Netzwerk abgewickelt werden.
- Da die Ergebnismengen durch effektive Filterung sehr klein sind, können komplexe Operationen im Postprocessing wie der Smith-Waterman Algorithmus auf günstiger Standard-PC Hardware durchgeführt werden.

Art der Suche	Anfrage	Anzahl der Hits	Ausführungszeit (sec)					Ethernet Bandbreite
			T_b	T_c	T_s	T_{SW}	T	
similarity	Line	5449	0,0	0,8	3,0	38,2	39	29,0 KB/s
similarity	Globin	329	0,0	0,8	3,0	1,6	3,8	1,7 KB/s
similarity	Prion	66	0,0	0,8	3,0	0,3	3,8	0,3 KB/s
similarity	Random	2	0,0	0,8	3,0	0,1	3,8	0,0 KB/s
pattern	OR1	20985	311	0,8	17,7	0,0	330	9,5 KB/s
pattern	OR2	13499	220	0,8	17,7	0,0	239	6,1 KB/s
pattern	OR3	17469	241	0,8	17,7	0,0	260	7,9 KB/s
pattern	OR4	24178	235	0,8	17,7	0,0	244	10,9 KB/s
pattern	OR5	21692	287	0,8	17,7	0,0	306	9,8 KB/s

Tabelle 3: Zusammenfassung der Testergebnisse, wobei vier verschiedene Sequenzen und fünf *olfactive receptor* Muster (OR) verwendet wurden. T_b ist die benötigte Zeit zum Broadcasten der neuen Konfiguration und fällt bei der Ähnlichkeitssuche weg. T_c ist die zur Rekonfiguration der Nodes benötigte Zeit, T_s die eigentliche Suchzeit und T_{SW} die für das Postprocessing benötigte Zeit. Da die Verarbeitung am Hostcomputer und die Suche in den Nodes sich überlappen, ergibt sich die Summe $T = T_b + T_c + max(T_s, T_{SW})$.

Zusätzliche steigt die Leistung von FPGA-Komponenten derzeit wesentlich schneller als bei normalen Mikroprozessoren. Zwar sind beide Arten von Chips dem Moore'schen Gesetz unterworfen, aber trotzdem steigt zumindest derzeit nach Aussage der Autoren die Leistungsfähigkeit der FPGAs schneller. Dies wird als weiterer Vorteil gegenüber PC Clustern gewertet, obwohl es sich dabei eher um einen zeitlich begrenzten Fakt handelt.

Die Nachteile des Systems liegen laut den Autoren bei der langsamen Netzwerkperformanz und der komplexen Implementierung neuer Applikationen. Erstere spielt im normalen Betrieb keine größere Rolle, lässt aber Updates der Datenbanken und Konfigurationen zu einer langwierigen Aufgabe werden; zweitere ist dagegen schon wesentlich schwerwiegender. Da die Implementierung neuer Applikationen oder nur die Anpassung fest einkodierter Schwellwerte detaillierte Hardwarekenntnisse erfordert, ist sie für den normalen Nutzer, zum Beispiel einen Biologen, zu komplex. Erweiterungen oder Anpassungen müssen somit immer durch einen erfahrenen Hardwaredesigner durchgeführt werden.

Bei genauer Betrachtung zeigt sich die Leistung des Systems eher als theoretische Laborergebnisse, die im praktischen Einsatz derzeit noch deutlich niedriger liegen dürften. So kann die Ähnlichkeitssuche nur Sequenzen mit maximal 160 Basenpaaren verarbeiten. Alle längeren Suchsequenzen müssen in mehreren Durchgängen abgearbeitet werden, was die Leistung enorm sinken lässt und wesentlich höhere Anforderungen an das Postprocessing und damit den Hostcomputer stellt. So liegt die Durchschnittslänge von Suchsequenzen nach Statistiken im Internet [12] bei ungefähren 315 Basenpaaren – es gibt also eine hohe Anzahl von Sequenzen die zwei, drei oder gar mehr Durchgänge erfordern.

Ein weiterer Kritikpunkt ist der implementierte Algorithmus, der gegenüber BLAST fehlerintoleranter ist. Da bei größeren voreingestellten Schwellwerten die Wahrscheinlichkeit, dass eine eigentlich valide Sequenz aufgrund einer Lücke oder Änderung gegenüber der Suchsequenz abgelehnt wird, exponentiell zunimmt, werden die Algorithmen mit zunehmender Genauigkeit im Gegensatz zum intuitiven Verständnis schlechter. Dies verstärkt den Effekt, dass für hochqualitative Resultate mehr Leistung im Postprocessing benötigt wird, noch weiter. Eine genaue Analyse und quantitative Aussagen dazu fehlen hier.

Bei der Mustersuche stellt sich das Bild ähnlich dar: auf der langsamsten Stufe (einfacher Automat) können nur ca. 98 % der Muster aus der PROSITE Datenbank implementiert werden. Es gibt also Muster von hoher Komplexität, die von diesem Cluster nicht verarbeitet werden können. Die Möglichkeit der Nachbearbeitung teilweiser Alignments im Postprocessing stellt wieder höhere Anforderungen an den Hostcomputer und wurde in diesem Paper auch nur kurz angesprochen, aber nicht genauer erläutert.

Abschließend kann man sagen, dass das Projekt sehr vielversprechend ist. Die Hauptziele, Geschwindigkeit und Kosteneffizienz, wurden erreicht, auch wenn dabei einige Einschränkungen in Kauf genommen werden müssen. Diese resultieren aber größtenteils in den beschränkten Ressourcen des verwendeten FPGAs - eine Variable, die sich mit der Zeit stetig bessert.

Literatur

[1] I. Tuomi: *The lives and death of Moore's law* First Mon. 7 (2002).

[2] S. Guyétant, M. Giraud, S. Derrien, L. L'Hours, S. Rubini, D. Lavenier, F. Raimbault: *Cluster of reconfigurable nodes for scanning large genomic banks* Par. Comp. 31(1), **73-96** (2005).

[3] T. Smith, M. Waterman: *Identification of common molecular subsequences* J. Mol. Biol. 147, **195-197** (1981).

[4] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman: *Basic local alignment search tool* J. Mol. Biol. 215(3), **403-410** (1990).

[5] Wikipedia (Englisch): *BLAST algorithm*, Stand 9.12.2005, 12:37 Uhr http://en.wikipedia.org/.../BLAST

[6] S. Altschul, T. Madden, A. Schäffer, J. Zhang, Z. Zhang, W. Miller, D. Lipman: *Gapped BLAST and Psi-BLAST: a new generation of protein database search programs* Nucl. Acids Res. 2, **3389-3402** (1997).

[7] W. Kent: *BLAT - The BLAST-Like Alignment Tool* Genome Res. 12(4), **656-664** (2002).

[8] C. Sigrist, L. Cerutti, N. Hulo, A. Gattiker, L. Falquet, M. Pagni, A. Bairoch, P. Bucher: *PROSITE: a documented database using patterns and profiles as motif descriptors* Brief Bioinform. 3, **265-274** (2002).

[9] B. Ma, J. Tromp, M. Li: *Patternhunter: faster and more sensitive homology search* Bioinform. 18(3), **440-445** (2002).

[10] D. Lavenier, S. Guyétant, S. Derrien, S. Rubini: *A reconfigurable parallel disk system for filtering genomic banks* ERSA'03, Engineering of Reconfigurable Systems and Algorithms, Las Vegas, Nevada, USA (2003).

[11] M. Giraud, D. Lavenier: *Linear encoding scheme for weighted finite automata* Lecture Notes in Computer Science 3317, **146-155** (2004).

[12] I. Rossi, P. Fariselli, P. Martelli, G. Tasco, R. Casadio: *Code porting and optimization of BLAST on a Beowulf Cluster* C.I.R.B. Biocomputing Unit, University of Bologna, Italy.