Fakultät Informatik Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Untersuchungen zur Implementierung von Genom-Alignmentalgorithmen auf modernen parallelen Plattformen

Belegverteidigung

Oliver Knodel - Oliver.Knodel@mailbox.tu-dresden.de

Dresden, 11.08.2010

01 Einleitung

Problemstellung

1. Neu gewonnene DNA-Sequenzen werden als erstes mit den bekannten Datenbanken abgeglichen, um Eigenschaften zu ermitteln
   - Inexakte Suche
   - Ergebnis ist die Position in der Datenbank
2. Exponentielles Wachstum der Datenbanken (mehrere Milliarden Basenpaare)

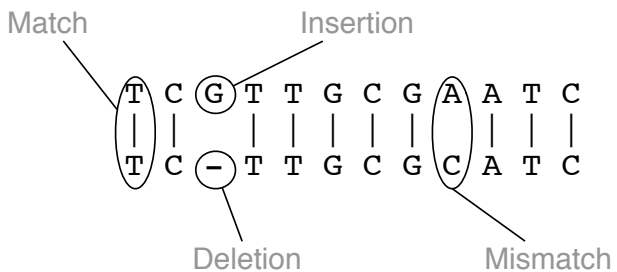
3. Neue kostengünstige und automatisierte Sequenzierungsverfahren haben in den letzten Jahren zum *Short Read Mapping Problem* [TS09] geführt
   - Kurze Sequenzen (*Reads*) von 30 - 100 Basenpaaren
   - Ein Durchlauf von 2-3 Tagen erzeugt mehrere Millionen Reads
   - Suche mit großen Computer-Clustern dauert mehrere Tage und solche Systeme sind nicht für jeden verfügbar

$\Rightarrow$ **Schnelle kostengünstige Lösungen erforderlich**

02 Alignment-Algorithmen

Alignment-Algorithmen passen zwei Sequenzen aneinander an und liefern einen Wert (*Score*), der etwas über ihre Ähnlichkeit aussagt

- Um einen hohen Score zu erreichen, werden *Mismatches* und *Gaps* eingefügt
- Die Algorithmen können für die inexakte Suche in Datenbanken eingesetzt werden
  - Ergebnis einer Datenbanksuche ist die Position, an der ein bestimmter Schwellwert für den Score überschritten wird
  - Schwellwert muss vor der Suche festgelegt werden

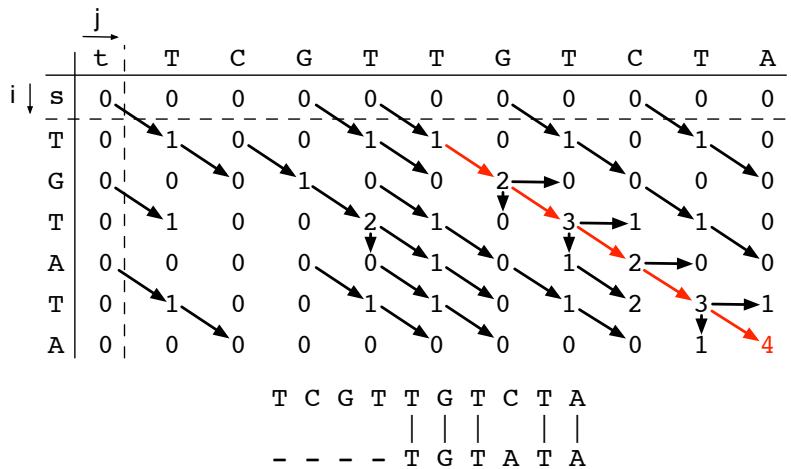


02 Alignment-Algorithmen

Smith & Waterman [Han06]

- Algorithmus zum lokalen Alignment
- Exaktes Verfahren (liefert immer optimales Alignment)
- Matrixfelder werden in Abhängigkeit voneinander berechnet
- Zeit- und Speicherkomplexität von $O(m \cdot n)$

$$M(i,j) = max \begin{cases} \underbrace{M(i-1,j) + g,}_{\text{Insertion}} \\ \underbrace{M(i,j-1) + g,}_{\text{Deletion}} \\ \underbrace{M(i-1,j-1) + p(s_i, t_j),}_{\text{Match / Mismatch}} \\ 0 \end{cases}$$

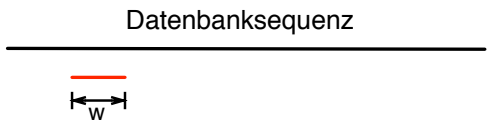


02 Alignment-Algorithmen

Basic Local Alignment Search Tool [Han06]

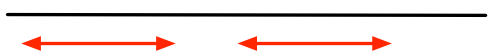
Heuristisches Verfahren zur Datenbanksuche für lange Sequenzen (bis zu 1.000 Basenpaare) bestehend aus drei Teilschritten:

1. Exakter Treffer der Länge *w* wird gesucht



2. Suche eines zweiten Hits mit maximalem Abstandes *A*

3. Hits werden mit Matches, Mismatches und Gaps ausgedehnt

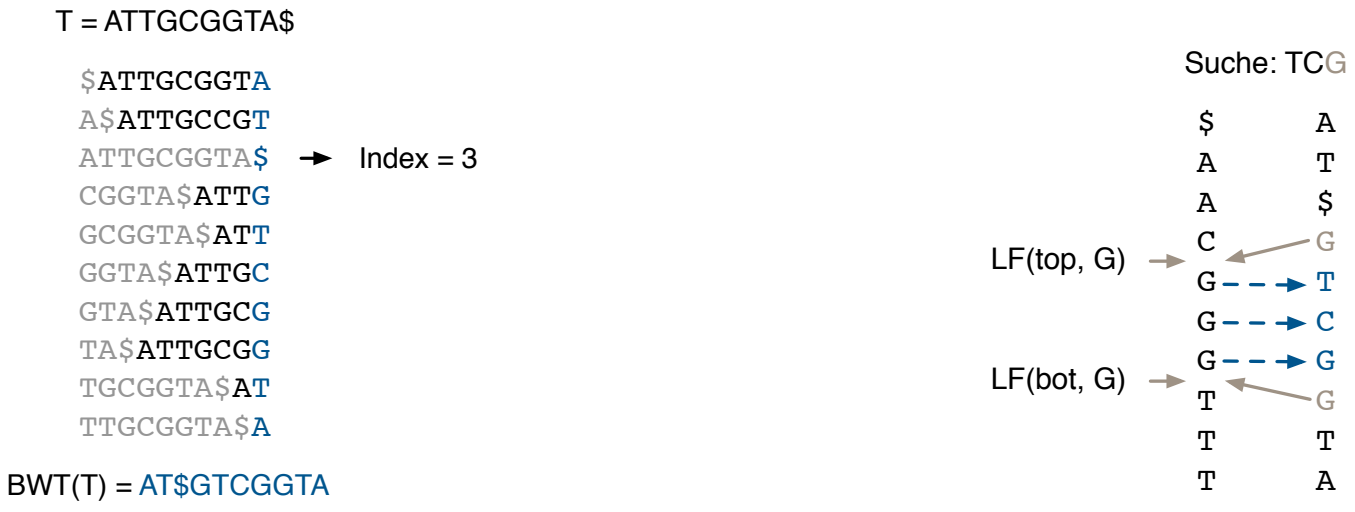


02 Alignment-Algorithmen

Short Read Mapper - Bowtie [TS09]

Gezielte Suche in Datenbanken mit Hilfe der Burrows-Wheeler-Transformation

- Hohe Geschwindigkeit, da nie die komplette Datenbank durchsucht wird
- Ausgelegt für Reads von 30 - 100 bp
- Einfügen von maximal 2 Mismatches über *Backtracking* (keine Gaps)



03 Plattformen

GPU - Graphics Processing Unit

- **S**ingle **I**nstruction **M**ultiple **T**hread (bzw SPMD)
- Hohe Speicherbandbreite (144 GByte/s)
- Floating-Point Leistung doppelter Genauigkeit von bis zu 515 GigaFLOPs
- Energieverbrauch von 200 - 250 Watt

[Niv10]

- Programierung mit CUDA (Compute Unified Device Architecture)
  - C/C++ Code mit CUDA Erweiterungen
  - Trennung von Programmierung und Grafikhardware

03 Plattformen

FPGA - Field Programmable Gate Array

- Konfigurierbare Hardware
- Programmierung mit Hardwarebeschreibungssprachen wie VHDL (Very High Speed Integrated Circuit Hardware Description Language)
- Taktrate von 100 - 200 MHz
- Energieverbrauch von 1 - 5 Watt

[Xil09]

03 Plattformen
Zusammenfassung

1. GPU
   - Vorteile
     - Viele Streaming Prozessoren mit hoher Taktrate
     - Hohe Speicherbandbreite
     - Einfache Erweiterung eines C-Programmes mit CUDA möglich
   - Nachteil
     - Hoher Energieverbrauch

2. FPGA
   - Vorteile
     - Direkte Umsetzung eines Algorithmus in Hardware
     - Geringe Taktrate, aber schnelle interne Logik und Verbindungsleitungen
     - Sehr geringer Energieverbrauch
     - Viele Kommunikationsschnittstellen (PCIe-x1, Gigabit-Ethernet)
   - Nachteil
     - Aufwändige Beschreibung der Hardware mit VHDL

04 Implementierung

Auswahl des Algorithmus

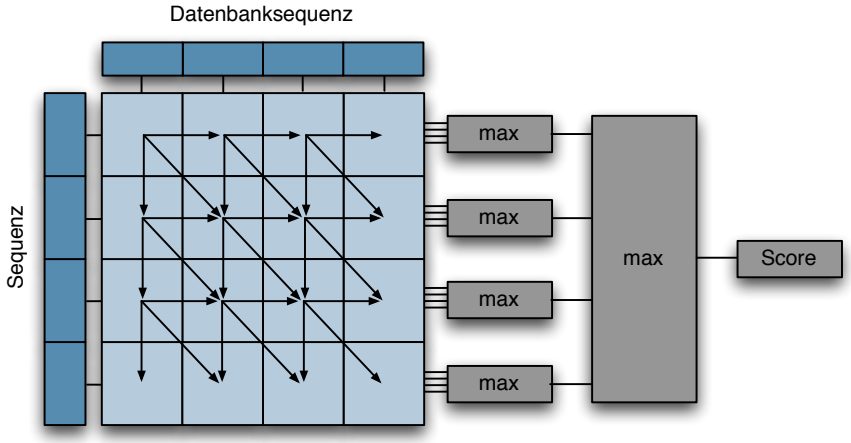
1. Smith & Waterman
   - Kann durch seine einfache Struktur einfach auf SIMD-Plattformen umgesetzt werden
   - Hoher Aufwand, wenn die gesamte Matrix berechnet wird

2. BLAST
   - Nicht effizient auf Short Read Mapping Problem erweiterbar
   - Unterschiedliche Phasen erschweren Umsetzung auf einer SIMD-Plattform

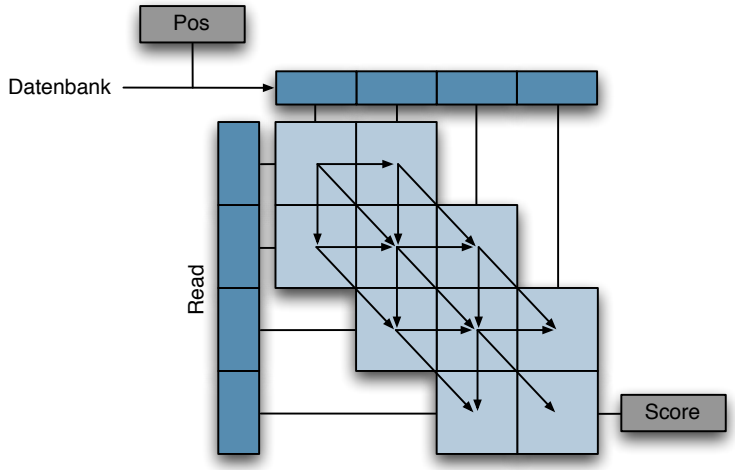
3. Bowtie
   - Hohe Geschwindigkeit
   - Eigentlicher Suchalgorithmus sehr schlecht parallelisierbar
   - Schlechtes Verhalten in Bezug auf Mismatches

$\Rightarrow$ **Smith & Waterman-Algorithmus**

Anpassung des Algorithmus an kurze Reads I



[LST07]



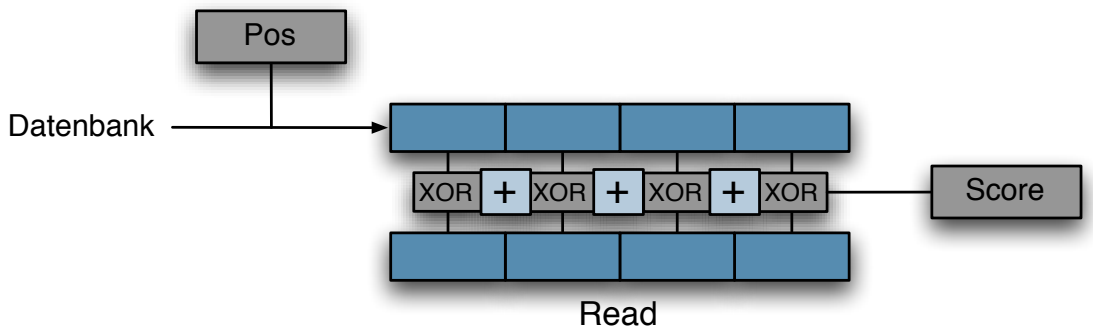
⇒ **Verkleinerung der Matrix auf direkte Umgebung der Diagonalen**

- Aufwand auf 4,94 % der kompletten Matrix mit 10.000 Feldern reduziert
- Needleman-Wunsch-Algorithmus für globales Alignment zur Berechnung
- Datenbank schrittweise um eine Position weiter schieben

Anpassung des Algorithmus an kurze Reads II

- Das Short Read Mapping Problem sieht keine Gaps vor
- Lediglich Mismatches werden akzeptiert

⇒ **Weitere Verkleinerung der Matrix auf die Diagonale**



04 Umsetzung

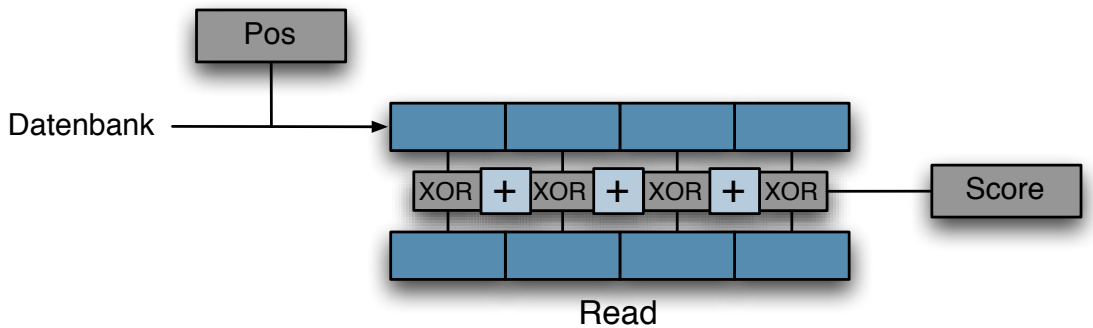
Auswahl der Plattform I

1. GPU
   - Für eine Umsetzung auf einer GPU verspricht der vorgestellte reduzierte Smith & Waterman-Algorithmus eine geringe Laufzeit und eine hohe Parallelität

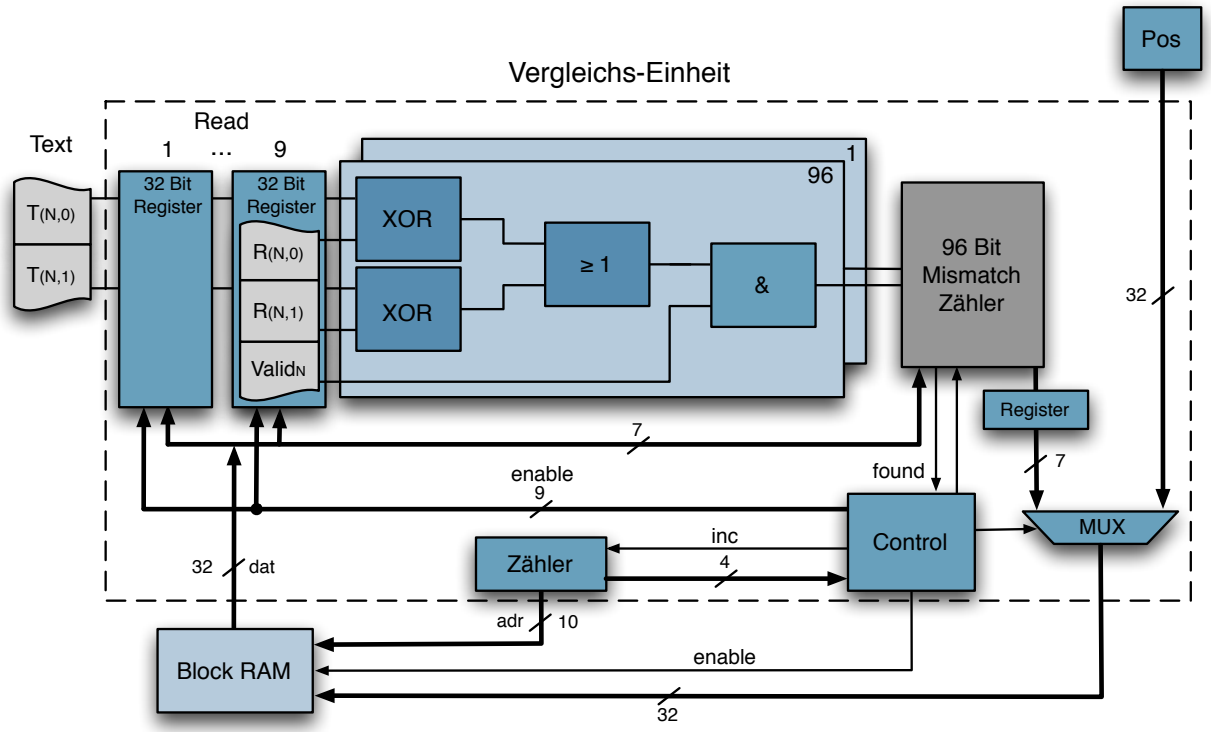
2. FPGA
   - Die Mismatches können über die komplette Länge des Reads mit einfacher Kombinatorik ermittelt und gezählt werden
   - Die Datenbank kann über Gigabit-Ethernet *gestreamt* werden und der Engpass Speicher wird umgangen

Auswahl der Plattform II

⇒ **FPGA-Ansatz mit erweitertem Intuitiven Suchalgorithmus verspricht hohe Geschwindigkeit bei hoher Parallelität durch geringen Hardwareaufwand**

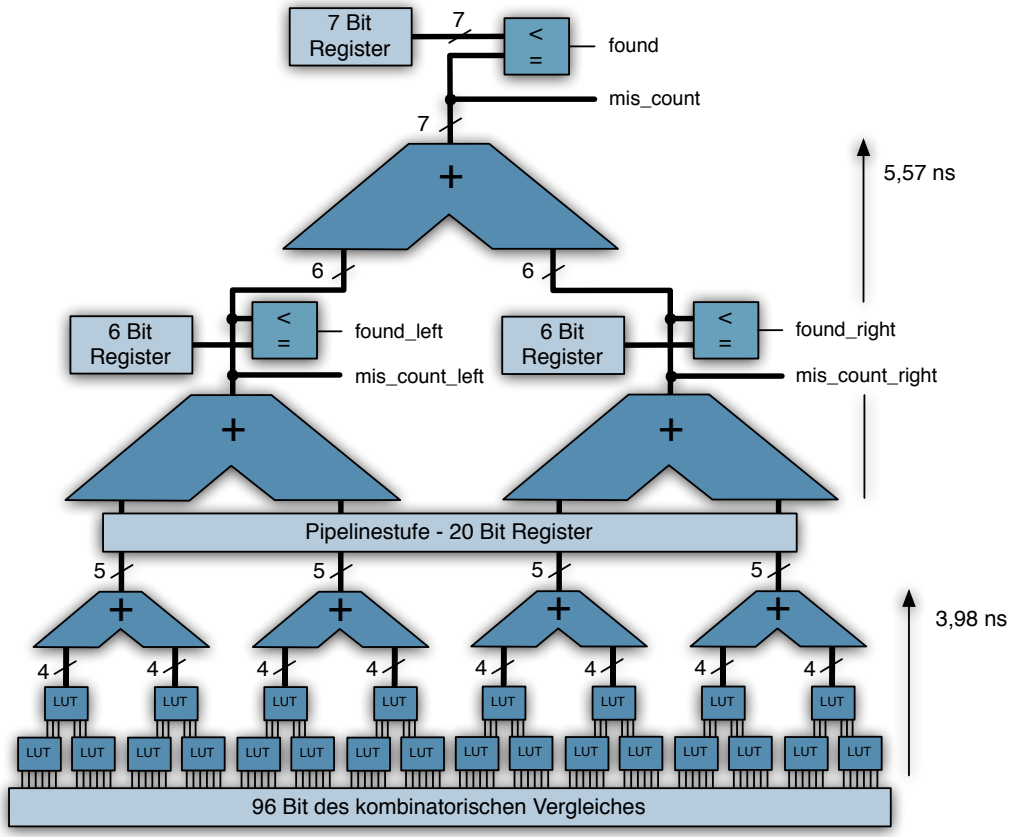


Implementierung - Vergleicher

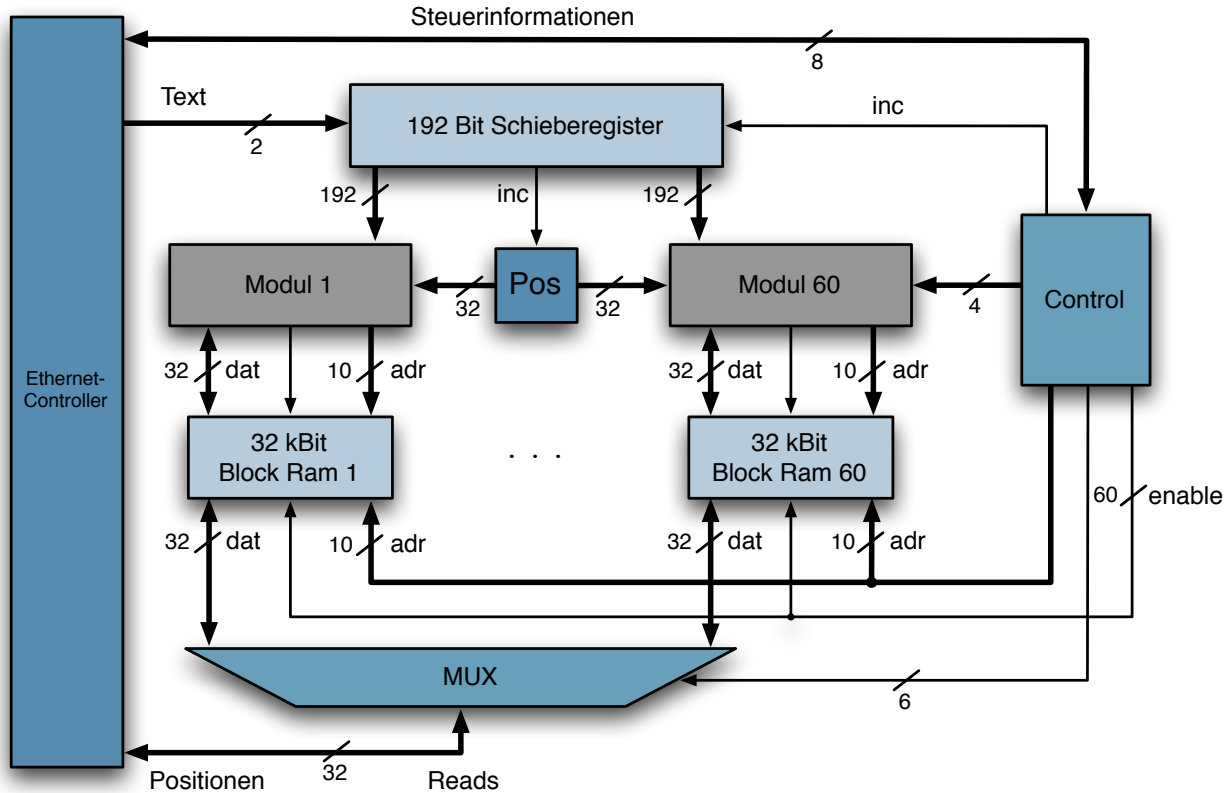


$$Vergleich_N = ((R_{(N,0)} \oplus T_{(N,0)}) \vee (R_{(N,1)} \oplus T_{(N,1)})) \wedge Valid_N$$

Implementierung - Zähler



Implementierung - Gesamtüberblick



05 Ergebnisse

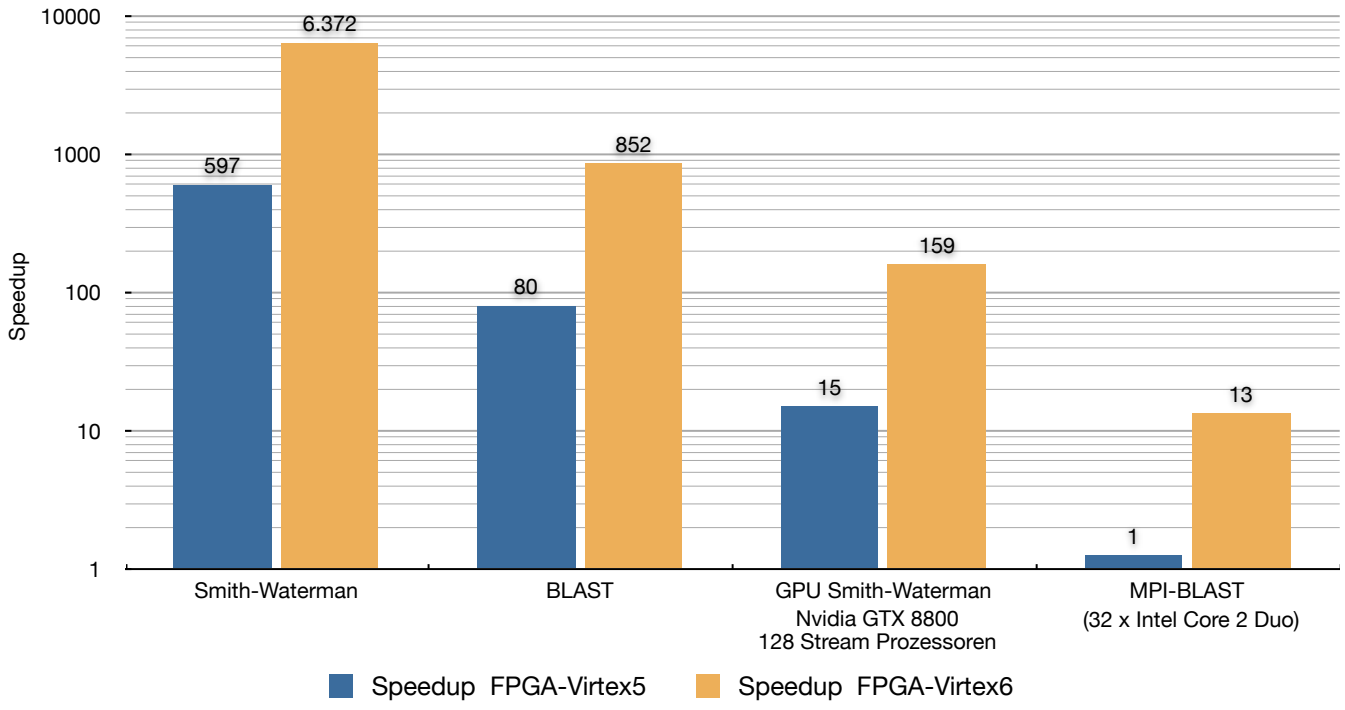
Implementierung - Erreichte Parallelität

FPGA	Virtex-5 LX50T	Virtex-6 LX240T
Parallele Einheiten	**75**	**400**
Register	27.119 (94 %)	143.824 (47 %)
LUTs	23.136 (80 %)	121.601 (80 %)
Block-RAM	60 (100 %)	400 (96 %)
Maximale Taktrate	100 MHz	200 MHz
Verlustleistung[1]	1,314 W	1,243 W
Vergleiche/s 96 bp	7.500.000.000	80.000.000.000
Vergleiche/s 48 bp	15.000.000.000	160.000.000.000
Erforderliche Datenrate	200 MBit/s	400 MBit/s

[1] Summe aus statischer und dynamischer Verlustleistung, ermittelt mit Xilinx ISE 12 XPower Analyzer

05 Ergebnisse

Theoretischer Speedup Datenbank: 122.655.632 bp - 1.000 Suchsequenzen (100 bp)



Testsystem: Intel Core 2 Duo 2,66 GHz, 3,8 GByte RAM, ein Thread
MPI-BLAST: Idealer Speedup - SWA-GPU [MV08]: Speedup für lange Sequenzen von 40

05 Ergebnisse

Kosten

⇒ 15 GPUs sind notwendig, um die Geschwindigkeit des FPGAs zu erreichen

Anschaffungskosten

1. Virtex-5 (ML505)
   - **1.195 €**
2. Nvidia GTX
   - 200 €
3. 15 Nvidia GTX
   - **3.000 €**

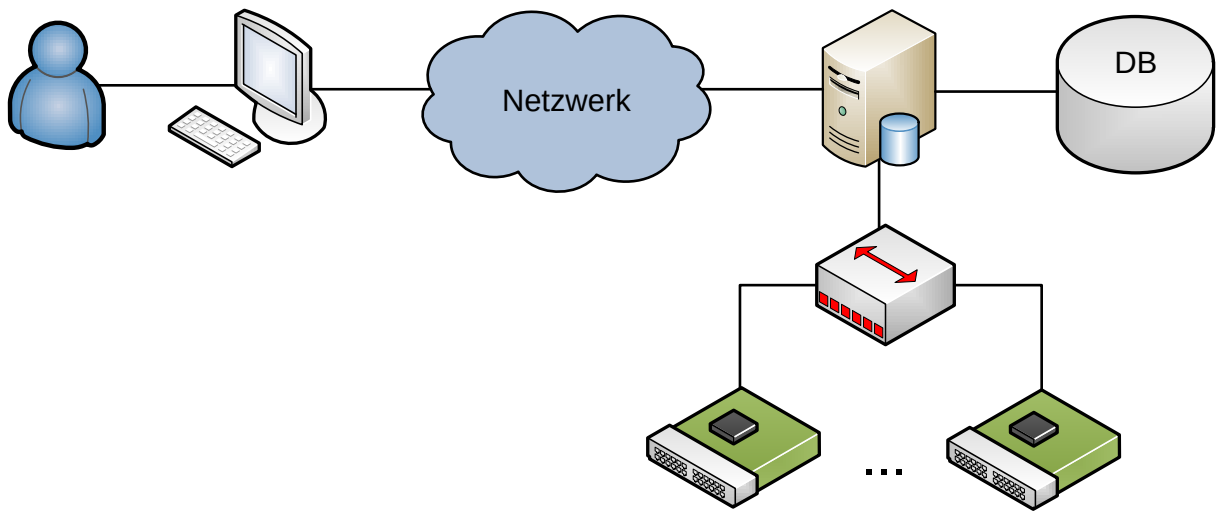
Betriebskosten (→ 8.760 Stunden)

1. Virtex-5 (ML505)
   - 1,31 Watt
   - 11,48 KWh
   - **2,30 €**
2. Nvidia GTX
   - 200,00 Watt
   - 1.927,00 KWh
3. 15 Nvidia GTX
   - 28.905,00 KWh
   - **5.781,00 €**

0,20 € je KWh (Drewag 2010)

06 Zusammenfassung und Ausblick

- Hohe Geschwindigkeit
- Exaktes und garantiertes Ergebnis für jeden Read
- Unabhängig von der Größe der Datenbank und Anzahl der Mismatches
- Geringe Einführungs- und Betriebskosten



07 Quellen I


[Niv10] Nvidia
Visual Computing Technologies
`http://www.nvidia.com` 2010


[MV08] Manavski, S. and Valle, G.
CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment
BMC bioinformatics 2008


[Han06] Hansen, Andrea
Bioinformatik: Ein Leitfaden für Naturwissenschaftler 2006


[KH10] Kirk, B. and Hwu, W.
Programming Massively Parallel Processors 2010


[Xi09] Xilinx
Virtex-5 FPGA User Guide
`http://www.xilinx.com/support/documentation/user_guides/ug190.pdf`
2009

07 Quellen II

[Xi10] Xilinx
Virtex-6 FPGA User Guide
`http://www.xilinx.com/support/documentation/user_guides/ug360.pdf`
2010

[TS09] Trapnell, C. and Salzberg, SL
How to map billions of short reads onto genomes
Nature biotechnology 2009

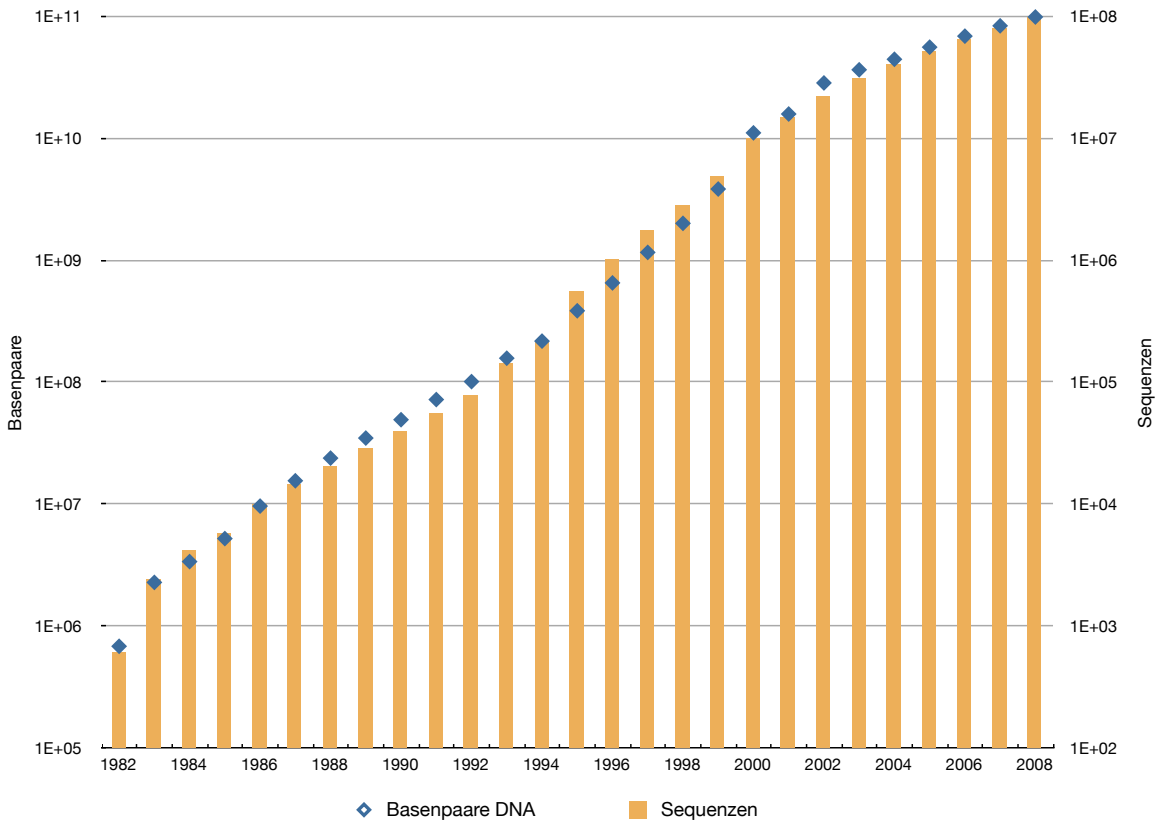
[Mar08] Mardis, E.R.
The impact of next-generation sequencing technology on genetics
Trends in Genetics 2008

[LST07] Li, I.T.S. and Shum, W. and Truong, K.
160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array(FPGA)
BMC bioinformatics 2007

»Wissen schafft Brücken.«

08 Anhang

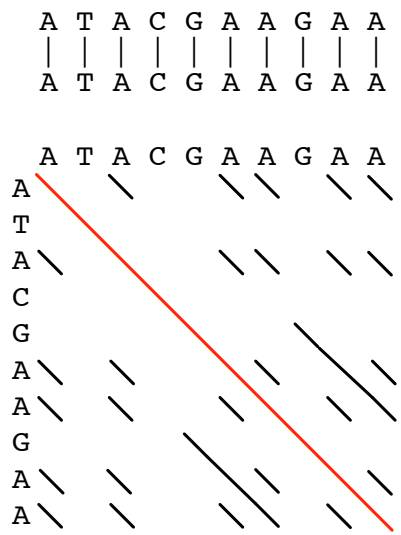
Datenbanken

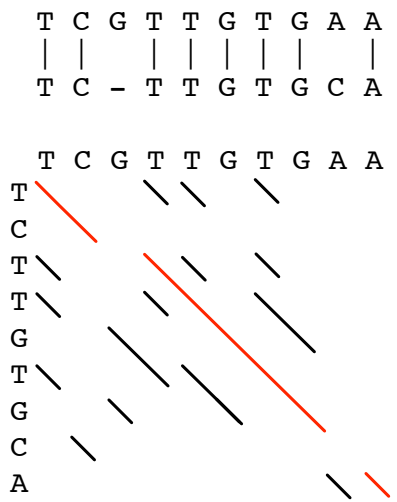


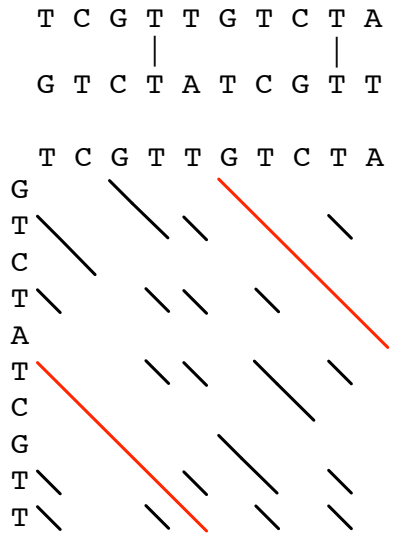
08 Anhang

Sequenzalignment - Dotplot

Dotplots liefern kein Alignment, sondern nur einen Überblick über die Ähnlichkeit der Sequenzen zueinander.

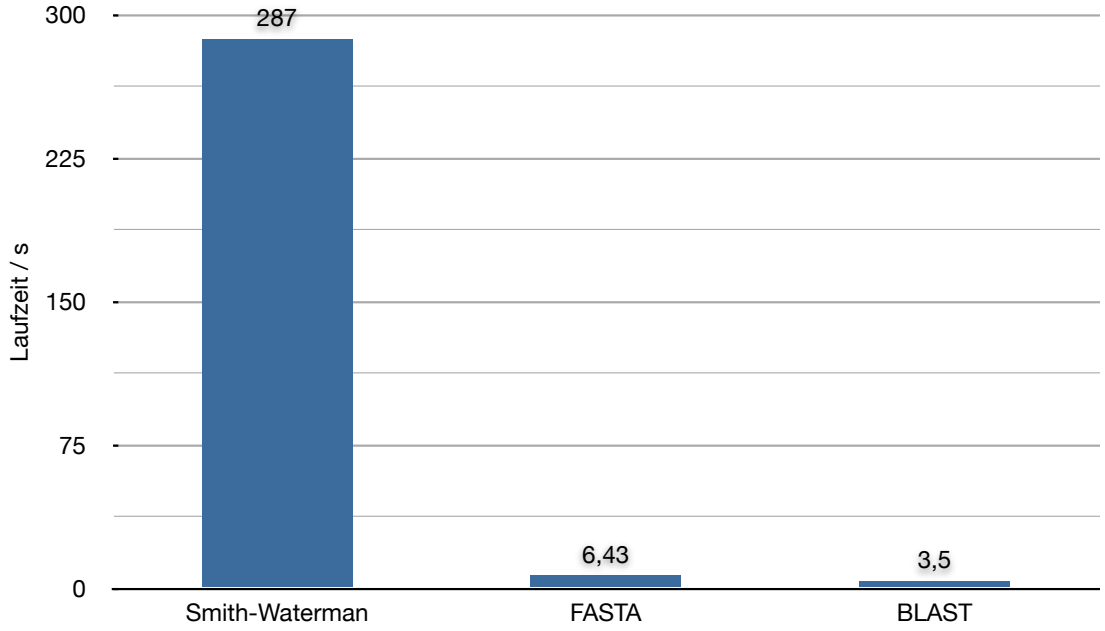






08 Anhang

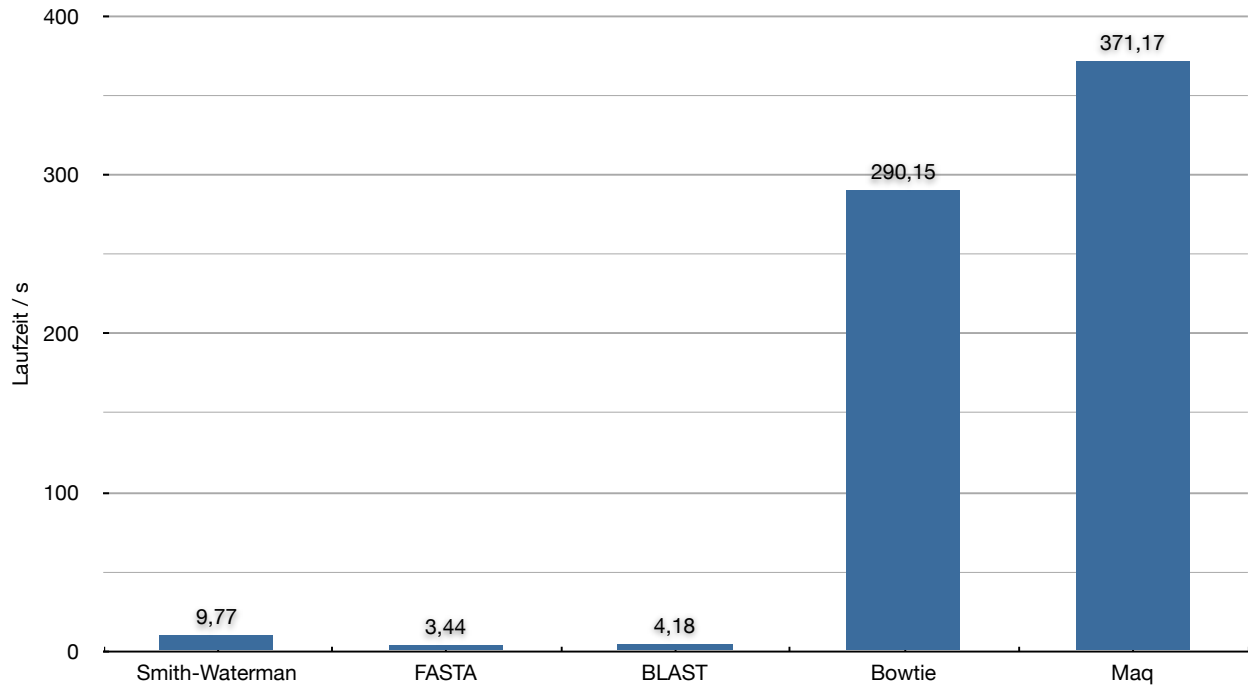
Ein Read (1.000 Basenpaare) Datenbank: 122.655.632 bp



Testsystem: Intel Core 2 Duo 2,66 GHz, 3,8 GByte RAM, ein Thread

08 Anhang

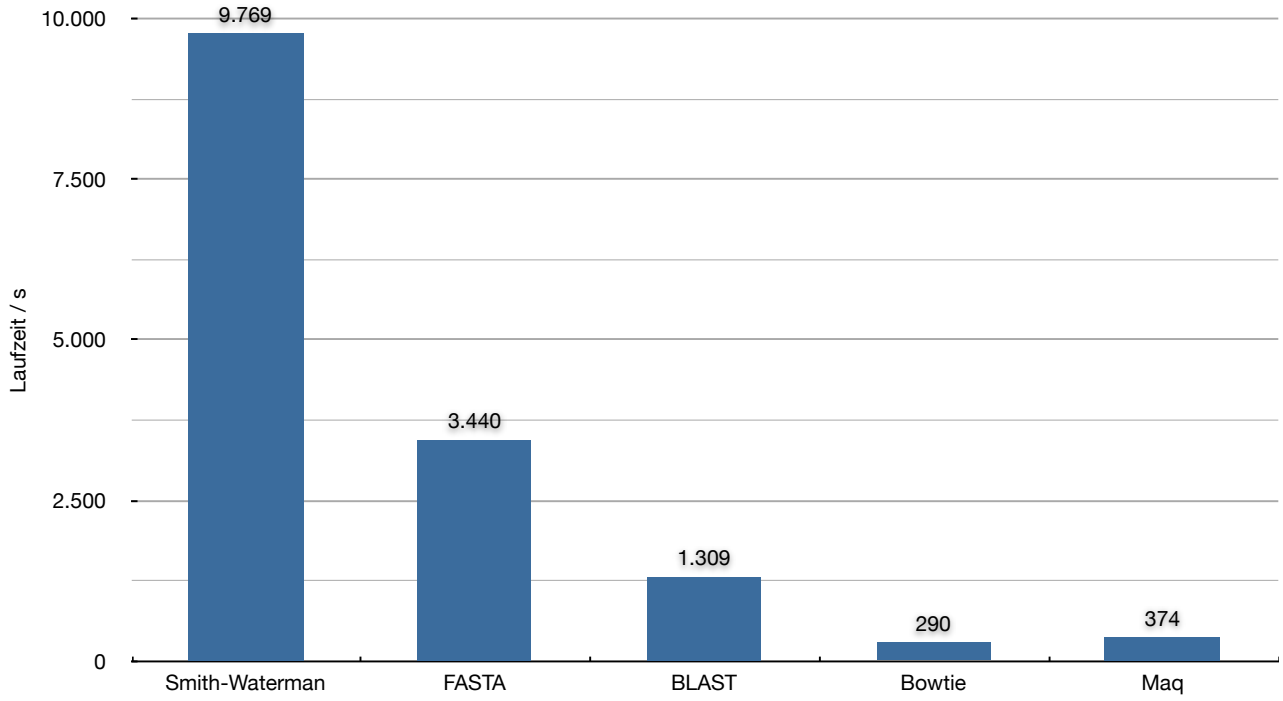
Ein Read (100 Basenpaare) Datenbank: 122.655.632 bp



Testsystem: Intel Core 2 Duo 2,66 GHz, 3,8 GByte RAM, ein Thread

08 Anhang

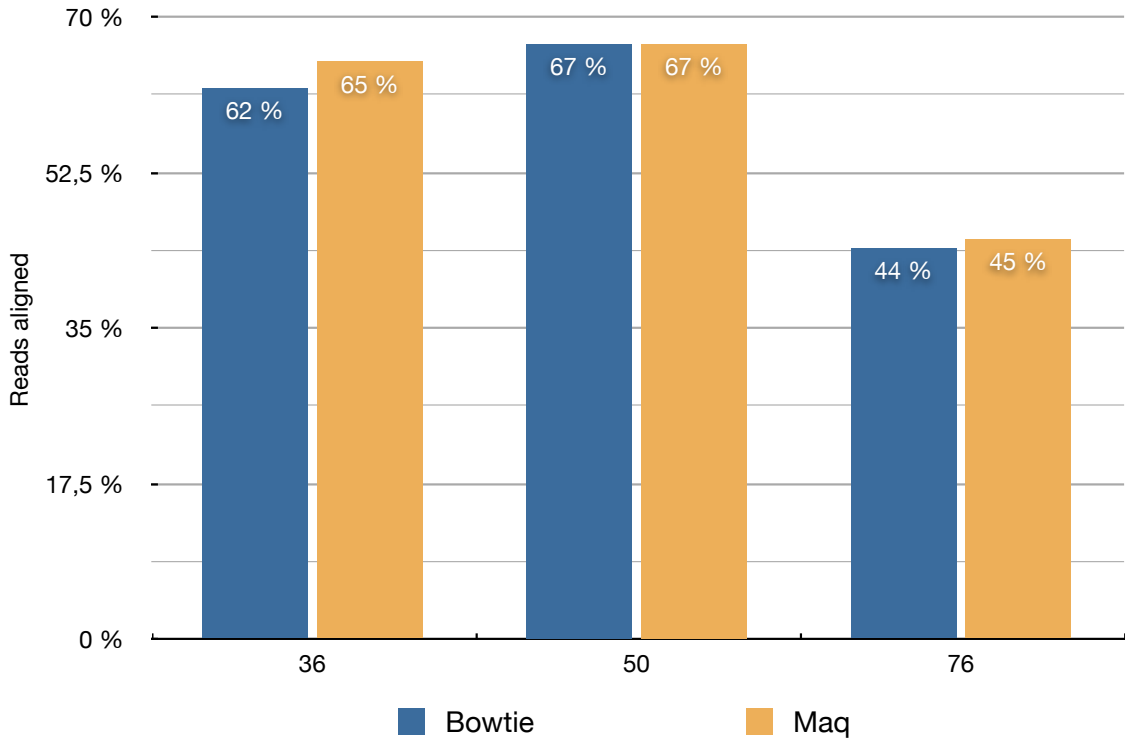
1.000 Reads (100 Basenpaare) Datenbank: 122.655.632 bp



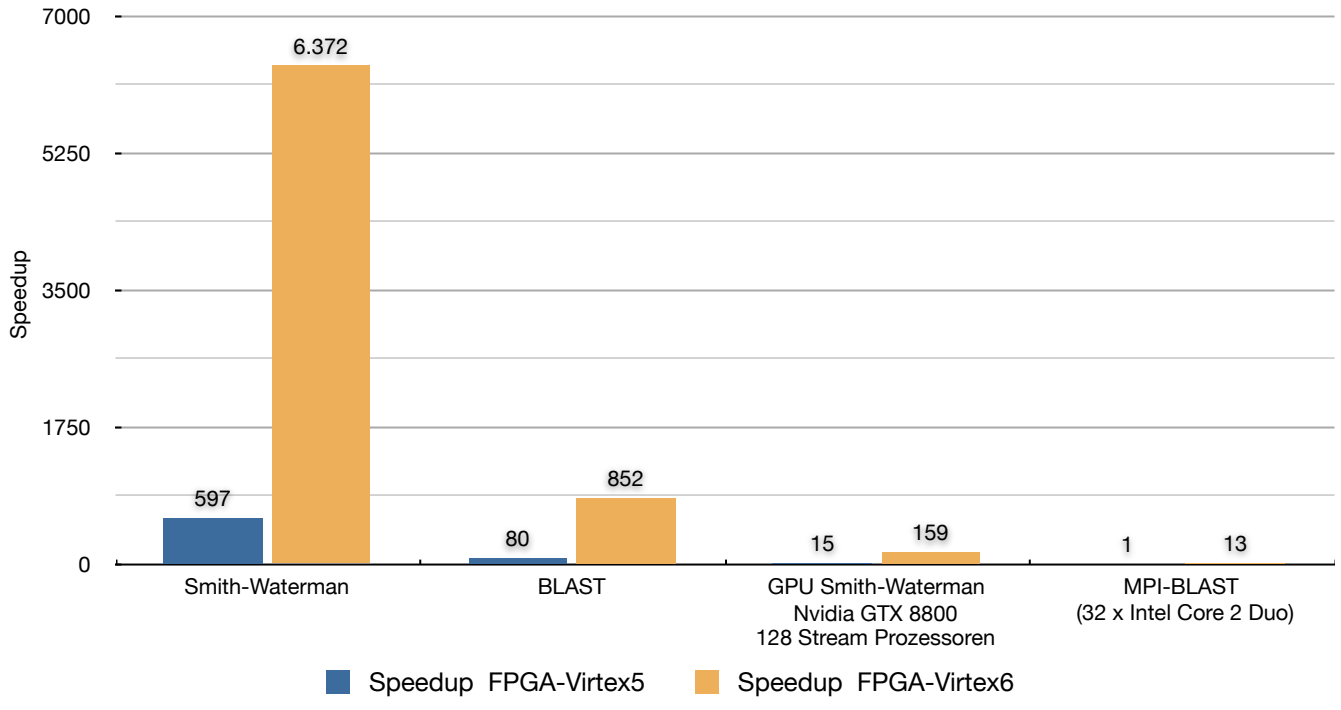
Testsystem: Intel Core 2 Duo 2,66 GHz, 3,8 GByte RAM, ein Thread

08 Anhang

Ergebnisse - Bowtie, Maq [TS09]



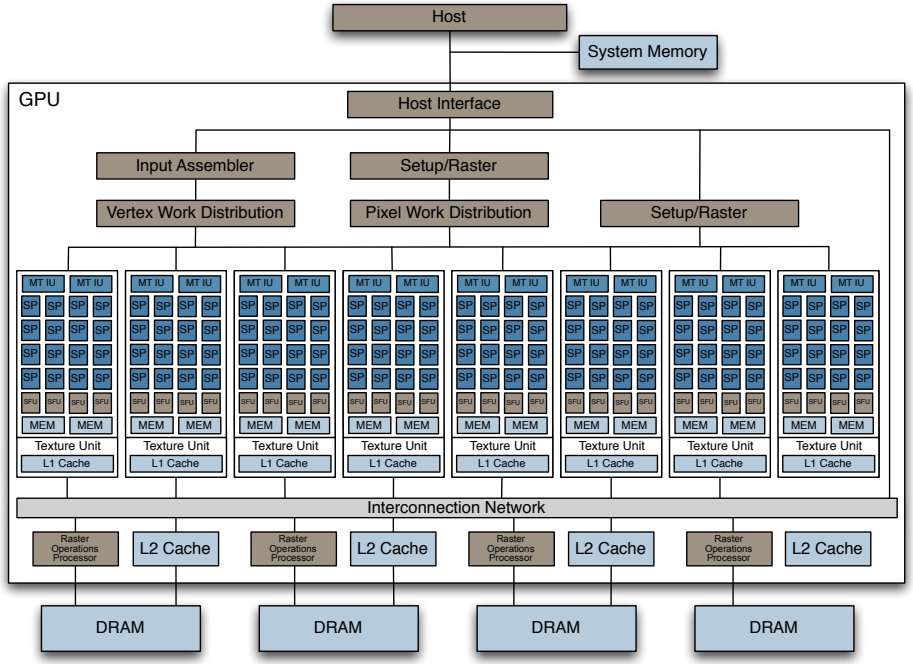
Theoretischer Speedup Datenbank: 122.655.632 bp - 1.000 Suchsequenzen (100 bp)



Testsystem: Intel Core 2 Duo 2,66 GHz, 3,8 GByte RAM, ein Thread
MPI-BLAST: Idealer Speedup - SWA-GPU: Speedup für lange Sequenzen von 40 [MV08]

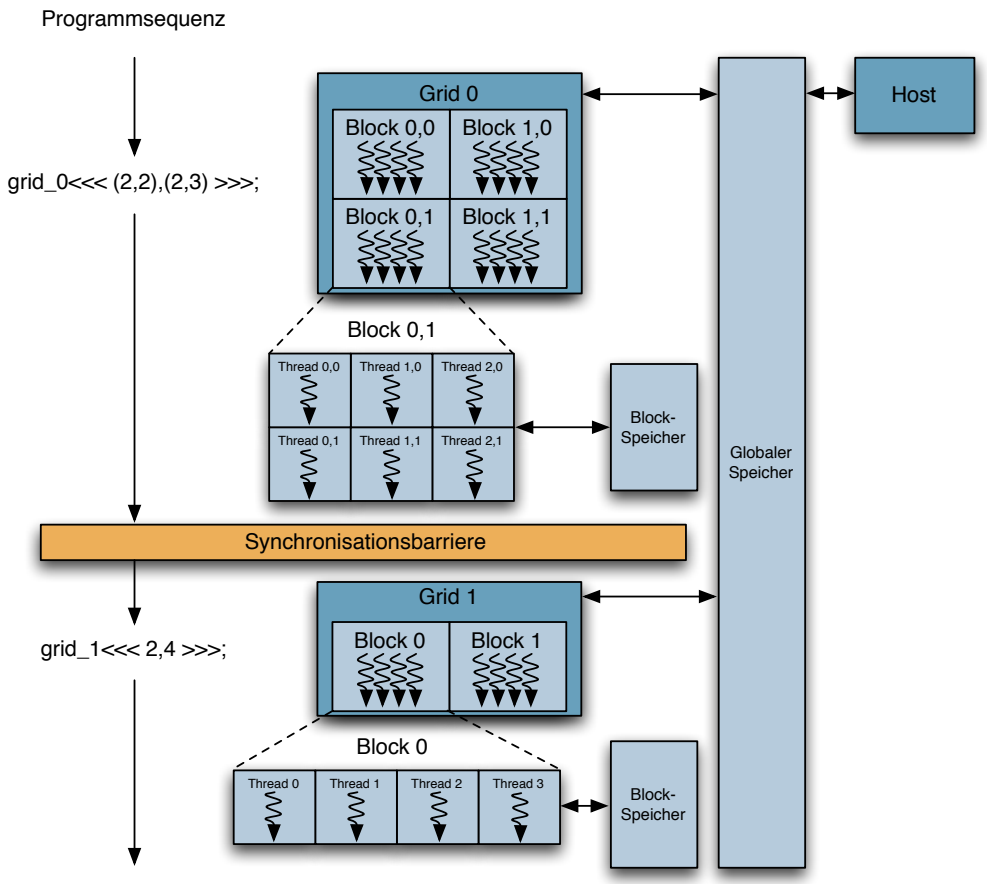
08 Anhang

GPU - Architektur der Tesla 8



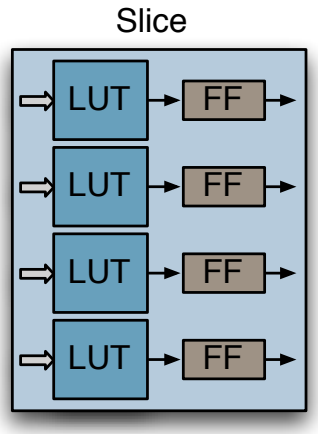
- Insgesamt 128 Streamprozessoren mit 1.500 MHz
- Jeder Streamprozessor kann 96 Threads verwalten
- Maximal sind 12.288 Threads möglich

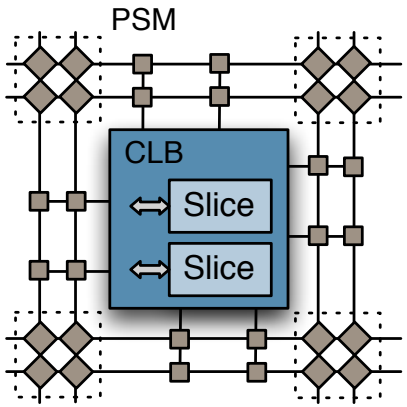
GPU - CUDA Programmiermodell

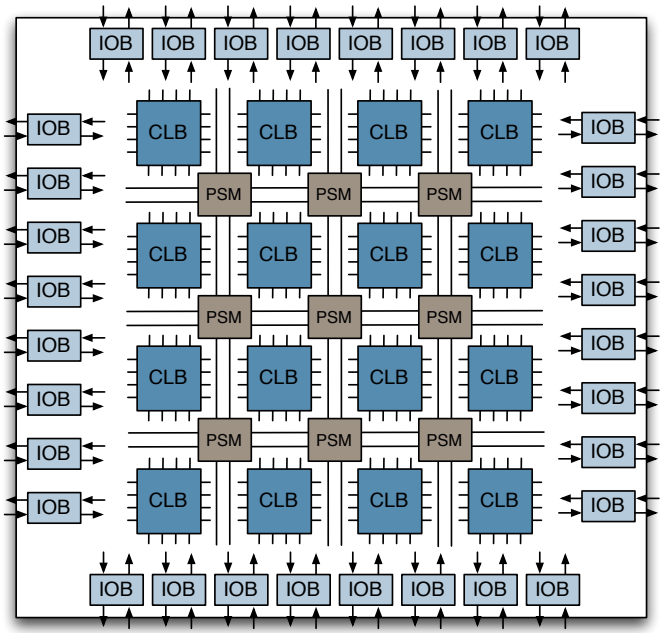


FPGA - Architektur Virtex-5

- 7.200 Slices
- 28.800 LUTs / Flip-Flops
- 2.160 KBit interner Dual Port Speicher (Block RAM)

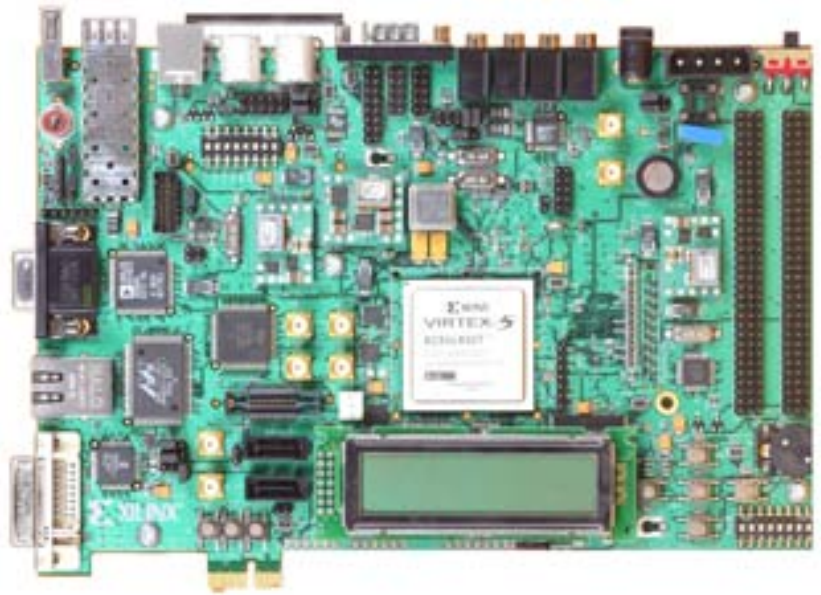






08 Anhang

FPGA - ML-505

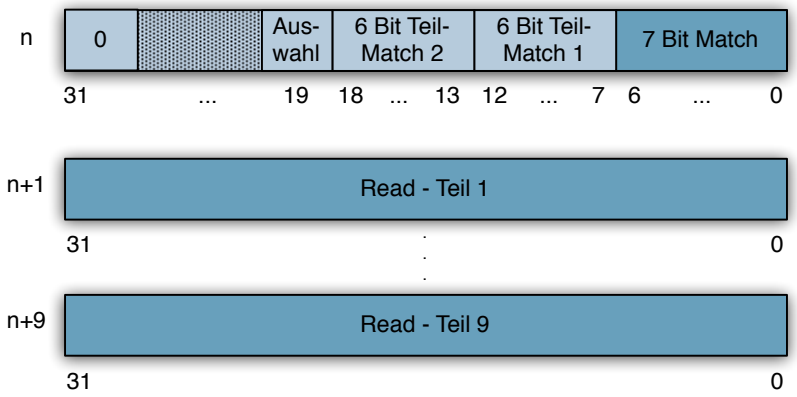


Mögliche Schnittstellen zur Kommunikation:

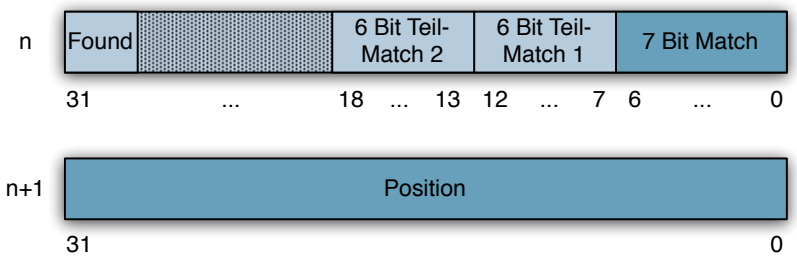
- Gigabit-Ethernet
- USB
- PCIe

Implementierung - Speicherverwaltung

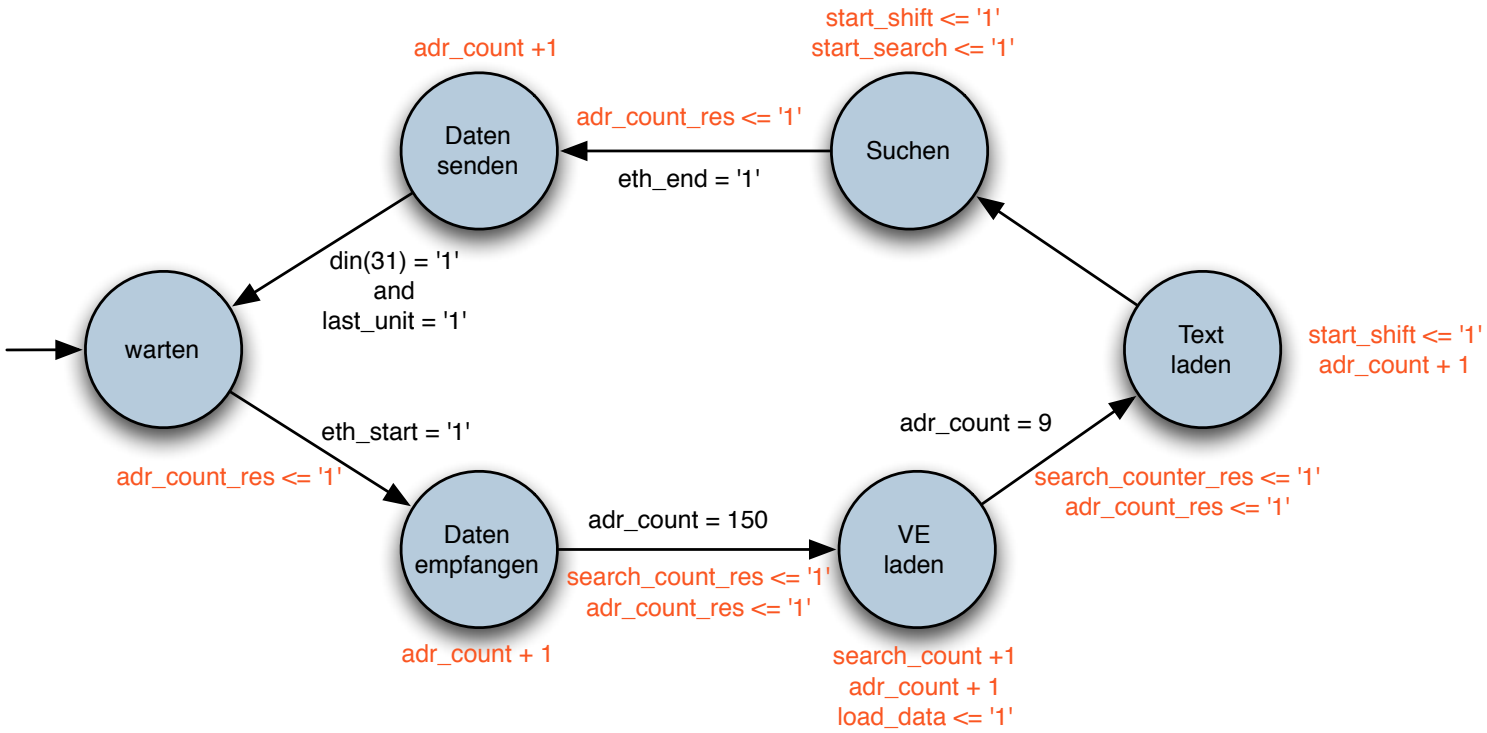
Ausgangsdaten:



Ergebnisdaten:

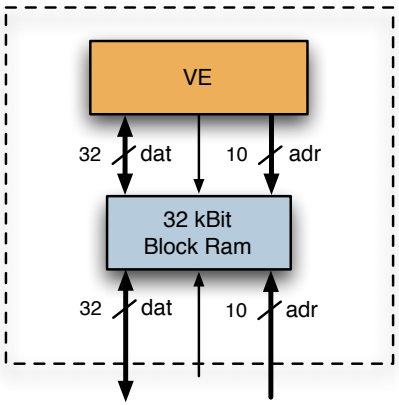


Implementierung - Zustände

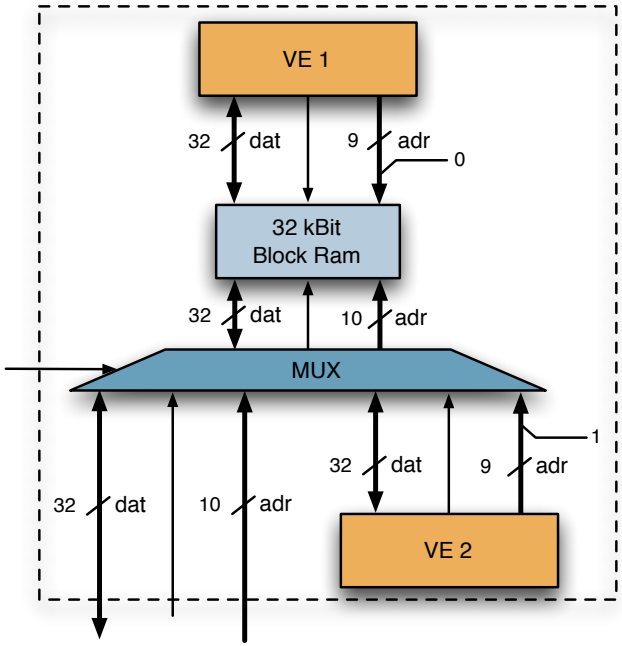


Implementierung - Module

Zur Erhöhung der Vergleichseinheiten unabhängig vom Block RAM können sich mehrere Einheiten einen Block RAM teilen



Modul mit einer VE



Modul mit zwei VEs

08 Anhang

GPU - CUDA Beispiel

Beispielberechnung $\vec{z} = a \cdot \vec{x} + \vec{y}$ in C:

```
void calc_serial(int n, float alpha, float *x, float *y) {
  for (int i; i<n; ++i)
  y[i] = alpha * x[i] + y[i];
}

//Funktionsaufruf
calc_serial(n, alpha, x, y);
```

in CUDA:

```
__global__
void calc_parallel(int n, float alpha, float *x, float *y) {
  int i = (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i < n) y[i] = alpha * x[i] + y[i];
}

//starte Kernel mit 256 Threads je Block
int nblocks = (n + 255) / 256;
calc_parallel<<<nblocks,256>>>(n, 2.0, x, y);
```