Smith-Waterman - Sequence Alignment mittels OpenCL

Laurence Bortfeld (l.bortfeld@gmail.com),
Wojciech Konitzer (w.konitzer@student.htw-berlin.de)
Prüfer: Sebastian Bauer

(Ausarbeitung)

***Abstract*—Abstract**

***Index Terms*—Smith-Waterman, OpenCL, Parallelität, Sequenz Aligment, DNA**

I. Einleitung

Seit einigen Jahren steigt die Taktfrequenz von Prozessoren (CPU) nicht weiter an, dies ist durch die hohe Wärmeabgabe bei hohen Taktfrequenzen des Prozessors bedingt. Die Entwicklung vom Mehrkernprozessoren erlaubt es durch Parallelisierung dies teilweise zu kompensieren. Mittlerweile ist nicht nur die Parallelisierung auf herkömmlichen CPUs von Bedeutung. Die rasante Entwicklung von Grafikarten-Prozessoren (GPU) macht diese für parallele Ausführung von Programmen immer interessanter. Verantwortlich ist die wesentlich höhere Anzahl an Prozessorkernen einer GPU im vergleich zu einer herkömmlichen CPU. Auch die die schneller ansteigende Leistung der GPUs im Bezug zu CPUs lassen der Grafikkarte für Parallelisierung mehr Bedeutung zukommen. [Bau13]

Diese Arbeit befasst sich mit der Parallelisierung eines Algorithmus für Sequenz Aligments in zwei Zeichenketten. Anwendung finden Algorithmen zur Bestimmung von Aligments zum Großteil in der Bioinformatik, um beispielsweise DNA-Sequenzen zu analysieren. Diese Arbeit betrachtet den Smith-Waterman Algorithmus, welcher das optimale lokale Alignment zweier Zeichenketten A und B ermittelt. Das verwendete Framework für die Parallelisierung auf der GPU ist OpenCL. Die Open Computing Language (OpenCL) definiert einen plattformübergreifenden Standart zum Ausführen von parallelen Anwendungen auf Mehrkern CPUs und GPUs [3]. Ziel dieser Arbeite ist es die Nebenläufigkeit des Smith-Waterman Algorithmus zu identifizieren und diesen mittels OpenCL auf der GPU zu parallelisieren. Vergleiche zwischen der seriellen und parallelen Version des Algorithmus geben Aufschluss darüber, ob eine effektive Parallelisierung des Algorithmus auf der GPU, unter Berücksichtigung der Implementierung, möglich ist.

II. Smith-Waterman Algorithmus

Der Smith-Waterman Algorithmus ist konstruiert um das optimale lokale Alignment zweier Zeichenketten oder Sequenzen zu bestimmen, somit ermittelt er den ähnlichsten Abschnitt in einer Zeichenkette. T.F. Smith und M.S. Waterman veröffentlichten den Algorithmus 1981 in dem Paper: *Identification of common molecular subsequences*. Wie der zuvor entworfene Algorithmus von Needleman & Wunsch (1970) wird mit Hilfe einer Matrix das Alignment berechnet. Es gibt eine viel Zahl von heuristischen Algorithmen, welche vor der Entwicklung des Smith-Waterman Algorithmus verfasst worden, jedoch waren diese für biologische Untersuchungen nicht hinreichend genug oder nicht interpretierbar. 1982 verbesserte Gotoh den Algorithmus vom Smith & Waterman. Der Ursprüngliche Algorithmus benötigte M^2N Schritte um das lokale Alignment zu erhalten, Gotoh reduzierte die benötigten Schritte auf MN, wobei M und N ($M \geq N$) die Längen der zu vergleichenden Zeichenketten bzw. Sequenzen sind. [SW81, Got82]

Bevor jedoch der Algorithmus vom Smith & Waterman beschrieben wird, soll der Unterschied zwischen lokalen und globalen Aligments geklärt werden. Lokale bzw. globale Alignments betrachten die zu untersuchenden Sequenzen unterschiedlich und ermitteln somit verschiedene Ergebnisse. Ein globales Alignment betrachtet das Alignment auf der gesamten Länge der Sequenzen (vgl. Listing 1). Hingen betrachtet das lokale Alignment nur ähnliche Abschnitte in einer Sequenz (vgl. Listing 1). Nun ist es möglich, dass mehrere lokale Alignments in einer Sequenz vorkommen, um das optimale lokale Alignment zu bestimmen, wählt ein Algorithmus das Alignment mir der höchsten Wertigkeit aus. Der Needleman & Wunsch Algorithmus ermittelt ein globales Alignment, indessen der Smith-Waterman Algorithmus ein optimales lokales Alignment bestimmt. [KS13]

Listing 1 Beispiel für globales und lokales Alignment

```
Global alignment:
A: A——DDAAAA–XXX————————A
B: AOPIODDAAAAZXXXASDASDASDAA

Local alignment:
A: DDAAAA–XXX
B: DDAAAAZXXX
```

A. Algorithmus

Der Smith-Waterman Algorithmus basiert auf dem Paradigma der dynamischen Programmierung. Hierbei ist "Programmierung" nicht im Sinne von schreiben von Code zu verstehen, dynamische Programmierung löst das Probleme durch das Ausfüllen einer Tabelle (Matrix). Wie auch bei bei

$$H(i,j) = max \begin{cases} H(i-1,j-1) + s(a_i,b_j) & \text{Match/Mismatch} \\ H(i-1,j) + s(a_i,-) & \text{Deletion} \\ H(i,j-1) + s(-,b_j) & \text{Insertion} \end{cases} \tag{1}$$

der Methode von “teile und herrsche” zerlegt dynamische Programmierung ein Probleme in viele leichter zu lösende Teilprobleme, deren Ergebnisse in einer Tabelle hinterlegt werden. Jedoch sind die Teilprobleme untereinander von einander Abhängig, da ihre Berechnungen bzw. Lösungen auf denen der Vorgänger beruhen. Generell lässt sich dynamische Programmierung auf Optimierungsprobleme anwenden. Solche Probleme bestehen aus einer Vielzahl von korrekten Lösungen, wohingegen nur eine optimale Lösung des Problems (Minima, Maxima) von Interesse ist. [Cor01] Gegeben sind zwei Zeichenketten bzw. Sequenzen $A = a_1 a_2 \ldots a_n$ und $B = b_1 b_2 \ldots b_m$. Die Ähnlichkeit zweier Elemente einer Zeichenkette (Buchstaben) sind durch die Funktion $s(a,b)$ definiert. Das entfernen von Elementen aus der Zeichenkette ist duck das Gewicht W_k bestimmt. Für das ermitteln von gleichartigen Segmenten in den Zeichenketten wird eine Matrix H mit den folgenden Werten initialisiert: $H_{k0} = H_{0l} = 0$ für $0 \leq k \leq n \wedge 0 \leq l \leq m$, wobei m, n die länge der Zeichenketten $|A|$ und $|B|$ sind. Die Werte in H ergeben sich aus den Operationen (siehe Formel 1) an der Stelle H_{ij} mit den Elementen a_i und b_j, hierbei ist $1 \leq i \leq n \wedge 1 \leq j \leq m$ zu beachten. Ist das Ergebnis einer Operation *negativ* ist es *Null* zu setzten. Um in der Zeichenkette das Segment zu finden, dass die größte Ähnlichkeit aufweist muss das Element in H gefunden werden, welches den größten Wert hat. Von diesem Element ausgehend könnenden die die nachfolgenden Elemente in einem Rückverfolgungsprozess[1] (Traceback) bestimmt werden. Der Traceback endet sobald ein Matrix Element *Null* ist. Dieser Prozess führ zu dem ähnlichsten Segmenten und zu dem optimalen Alignment der Zeichenketten A und B.[SW81]

B. Beispiel

Im Folgenden soll ein Beispiel gegeben werden, welches den Algorithmus verdeutlichen soll. Anhand der Zeichenketten $A =$ ANANAS und $B =$ BANANE, wobei die Parameter für den Vergleich wie wie folgt gewählt sind:

$$s(\text{match}) = 2 \wedge s(a,-) = s(-,b) = s(mismatch) = -1$$

Die Abbildungen Fig. 1(a) und 1(b) zeigen die Initialisierung der Matrix M sowie die Berechnung der jeweiligen Elemente von M durch die vorher festgelegten Operationen aus der Formel 1. Die Pfeile in der Abbildung Fig. 1(b) zeigen auf das Element aus dem sich das Element an der Stelle H_{ij} ergibt.

Ist die Matrix wie in der Abbildung Fig. 1(b) ausgefüllt, kann mittels des Tracebacks, ausgehend von dem Element mit dem höchsten Wert, die ähnlichste Sequenz aus der Zeichenkette A und B bestimmt werden (siehe Fig. 2). Aus dem

[1]Der Weg aus dem sich das Element im Vergleich mit den Operationen aus Formel 1 ergeben hat.

/	–	A	N	A	N	A	S
–	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0

(a) Initialisierung der Matrix M.

/	–	A	N	A	N	A	S
–	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1
N	0	1	4	3	4	3	2
A	0	2	3	6	5	6	5
N	0	1	4	5	8	7	6
E	0	0	3	4	7	7	6

(b) Berechnen der Elemente von M mit Operationen aus der Formel 1.

Fig. 1: Initialisierung und Berechnung der Matrix M.

Traceback ergibt sich somit ein Alignment für A und B von *ANAN*.

C. Serieller Ansatz

Die Implementierung des seriellen Ansatzes hat im Kern zwei Matrizen. Die erste Matrix H enthält die Werte, die aus den Ergebnissen der Operationen und kann sich wie die Matrix aus aus dem Beispiel II-B vorgestellt werden. Die zweite Matrix M ist für das Traceback notwendig. Sie dient für die Speicherung aus welchem Element sich das Element Hij zusammensetzt. Der Pseudocode 1 soll die Implementation der seriellen Version des Smith-Waterman Algorithmus verdeutlichen. Die beiden ersten Zeilen initialisieren die Matrix für die Berechnungen des Alignments und die Matrix für das Merken aus welchem Element sich das aktuelle Element ergibt. Das ausfüllen der Matrix mittels der Operationen aus Formel 1 findet in den Zeilen 3-8 statt. Zeile 6 wendet die Operationen auf das aktuelle Element H_{ij} an. In der darauffolgenden Zeile findet das Speichern der benutzen Operation statt, aus welcher sich der gegangene Pfad im Traceback rekonstruieren lässt. Die in den Zeilen 9-11 definierten Variablen dienen für den Traceback. *Result*$_a$ und *Result*$_b$ halten die im Traceback

/	–	A	N	A	N	A	S
–	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1
N	0	1	4	3	4	3	2
A	0	2	3	6	5	6	5
N	0	1	4	5	8	7	6
E	0	0	3	4	7	7	6

Fig. 2: Traceback in M um das optimale lokale Alignment zu bestimmen.

ermittelten Sequenzen in den Zeichenketten A und B. *Current* speichert den aktuellen Wert, $Next$ den Folgewert, welche mittels M bestimmt werden kann. In den Zeilen 12-25 wird das optimale lokale Alignment mittels H und M ermittelt und die ähnlichsten Segmente zurückgegeben.

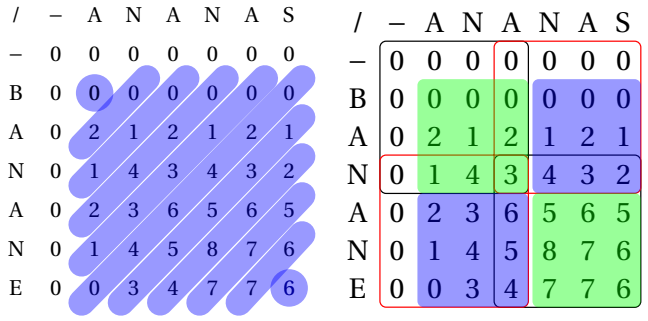
Pseudocode 1 Implementierung Smith-Waterman serieller Ansatz

```
Require: A, B ∈ [A − Z]
 1: H ← InitMatrix()
 2: M ← InitMatrix()
 3: for all 1 ≤ i ≤ n do                          ▷ Scoring the matirx
 4:     for all 1 ≤ j ≤ m do
 5:         H_ij ← FindMaximumWith(a_i, b_j)
 6:         M_ij ← MemomorizeMaxDirection()
 7:     end for
 8: end for
 9: Result_a, Result_b
10: Current ← MaximumValueIn(H)
11: Next ← GetNextWith(M, Current)
12: while Current ≠ Next ∧ Next ≠ 0 do            ▷ Traceback
13:     if Next_i = Current_i then                ▷ Deletion in A
14:         Result_a ← '-'
15:     else                                      ▷ Match/Mismatch in A
16:         Result_a ← A[Current]
17:     end if
18:     if Next_j = Current_j then                ▷ Deletion in B
19:         Result_b ← '-'
20:     else                                      ▷ Match/Mismatch in B
21:         Result_b ← B[Current]
22:     end if
23:     Current = Next
24:     Next ← GetNextWith(M, Current)
25: end while
26: return Result_a, Result_b
```

D. Paralleler Ansatz mittels OpenCL

Für eine parallele Implementierung des Smith-Waterman Algorithmus ist das Erkennen der Abhängigkeiten der Operatoren, welche in der Formel 1 definiert sind, notwendig. Aus den Indizes der benutzen Operatoren ($i-1, j-1$, $i-1, j$ und $i, j-1$) um auf Elemente aus H zuzugreifen geht hervor, dass die Operationen von Vorgängerwerten in der Matrix abhängen. Dies macht es nicht möglich H Zeile für Zeile parallel auszufüllen. Eine andere Herangehensweise ist es die Matrix in Antidiagonalen[2] zu berechnen (siehe Fig. 3(a)). Die Herausforderung hier bei ist, dass die Längen der Antidiagonalen in Abhängigkeit von n und m ändern. Somit müssen die Antidiagonalen dem entsprechend berechnet werden. Ein weiteres Problem bei diesem Ansatz ist, dass es zu einem ohne Overhead bei der parallelen Ausführung kommet, da die jeweiligen Antidiagonalen mit den Elementen von denen Sie Abhängen auf die entsprechenden Prozessoren verteilt werden müssen. Eine bessere Vorgehensweise ist es die Matrix H in mehrere Blöcke zu unterteilen (siehe Fig. 3(b)). Jeder Block ist nur von seinem Vorgänger abhängig. Die schwarzen und roten Rahmen in der Abbildung Fig. 3(b) zeigen den gesamten Block der für eine Berechnung betrachtet wird. Die grün und blau ausgefüllten Blöcke sind die Werte die in einem Block berechnet werden. In der Abbildung Fig. 3(b) ist zu erkennen, dass sobald der erste grüne Block berechnet ist, lassen sich die beiden nachfolgenden blauen Blöcke voneinander unabhängig berechnen. Zu Beachten ist, dass für jede Berechnung eines Blockes die erste horizontale und vertikale Zeile der vorigen Berechnung mitgeführt werden muss. Dies führt aber zu einem geringeren Overhead als die Herangehensweise mit Antidiagonalen. [MCU+01]

[2]Die Addition von i und j gibt pro Antidiagonale immer den selben Wert.



(a) Jedes Element in einer Antidiagonalen kann parallel berechnet werden.

(b) Jede Submatrix kann pro Antidiagonale (grün, blau) parallel berechenet werden.

Fig. 3: Parallele Ausführung des Smith-Waterman Algorithmus mittels Antidiagonalen. Antidiagonale müssen seriel berechnete werden. Jedoch sind ihre Element voneinander unabhängig

1) OpenCL:

2) SimpleCL:

3) Implementation:

III. DNA

Die DNA (Desoxyribonukleinsäure) ist ein Biomolekül und ist Bestandteil jedes Lebewesen und Viren. Es besteht aus vielen Bestandteilen, den sogenannten Nukleotiden. Jedes Nukleotid besteht aus Phosphorsäure bzw. Phosphat und Zucker (Desoxyribose) sowie einer einer Base. Bei der Base kann es sich um Adenin (A), Thymin (T), Cytosin (C) oder Guanin (G) handeln. Die Phosphorsäure und der Zucker sind immer gleich und bilden den Strang des DNA Moleküls. Dabei bilden immer zwei Nukleotide anhand ihrer Basen ein Basenpaar. Es können jedoch nur Basenpaare aus Adenin und Thymin oder Cytosin und Guanin gebildet werden (siehe Abbildung 4). Anhand der Komplexität der DNA Sequenz, die beim Menschen aus 3.101.788.170 Basenpaaren besteht, erfolgt die DNA Sequenzierung abschnittsweise. [1]

A. Aligments in der DNA

Der Vergleich (Alignment) von DNA Sequenzen spielt eine wichtige Rolle in der Forschung, Medizin, Forensik und Bioinformatik. Es ist mit Hilfe von Algorithmen, wie dem Smith-Waterman Algorithmus möglich zwei DNA Sequenzen miteinander zu vergleichen. Stellt man eine biologische Sequenz als einen eindimensionalen Zeichenkette dar, so kann

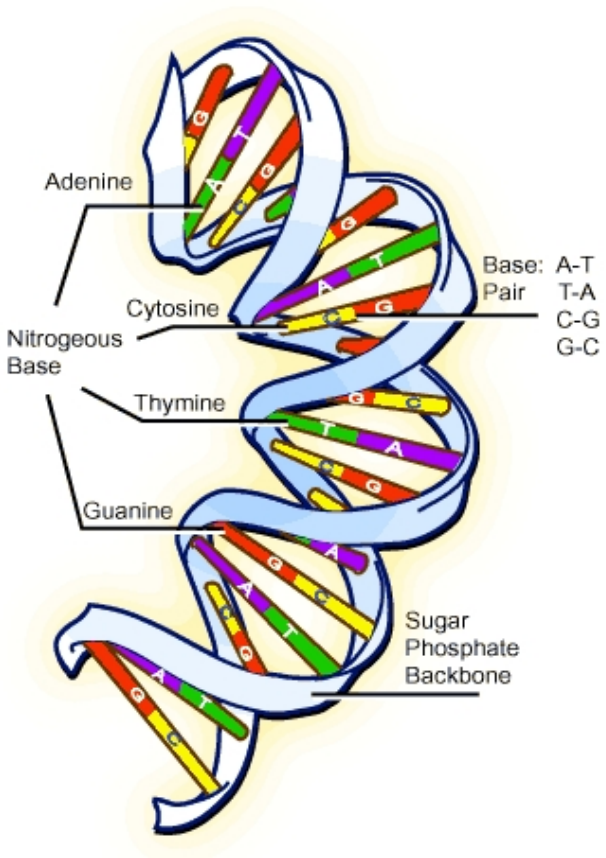


Fig. 4: Strukturmodell eines DNA-Moleküls [2]

der Vergleich der beiden Sequenzen als zeichenweiser Vergleich von diesen Zeichenketten verstanden werden. Dazu werden als Zeichen die Abkürzungen der Basenpaare (AT, TA, CG, GC) verwendet. Ein sehr kurzes Beispiel einer Sequenz könnte, wie folgt sein: “CGCGATCGATCGTACG”. Ziel ist es den Grad Ähnlichkeit bzw. Unähnlichkeit zu bestimmen.

In der Forensik können DNA Sequenzen mit einer vorgegebenen Sequenz verglichen werden, um Täter zu identifizieren. In der Forschung können z.B. Spezies mit anderen Spezies oder defekte DNA Sequenzen mit korrekten DNA Sequenzen verglichen werden. [1]

IV. Ergebnisse

V. Zusammenfassung

VI. Ausblick

Literatur

[Bau13] Bauer, Sebastian: *Parallel Systems: Mehrkern-Prozessoren*. Presented as Lecture SoSe 2013, 2013

[Cor01] Cormen, Thomas H.: *Introduction to algorithms*. 2. ed. Cambridge, Mass. [u.a.] : Cambridge, Mass. [u.a.] : MIT Press [u.a.], 2001

[Got82] Gotoh, Osamu: An Improved Algorithm for Matching Biological Sequences. In: *J. Mol. Biol*, 1982, S. 705–708

[KS13] Kohlbacher, Oliver ; Schmid, Steffen: *Bioinformatik für Lebenswissenschaftler: Paarweises Alignment – Teil II*. Presented as Lecture SoSe at Eberhard Karls Universität Tübingen, 2013

[MCU+01] Martins, W.S. ; Cuvillo, J.B. D. ; Useche, F.J. ; Theobald, K.B. ; Gao, G.R.: A MULTITHREADED PARALLEL IMPLEMENTATION OF A DYNAMIC PROGRAMMING ALGORITHM FOR SEQUENCE COMPARISON. In: *Communications of the ACM* (2001)

[SW81] Smith, T.F. ; Waterman, M.S.: Identification of Common Molecular Subsequences. In: *J. Mol. Biol*, 1981, S. 195–197

Internetquellen

[1] Wikipedia. Desoxyribonukleinsäure. Website, 2013. http://de.wikipedia.org/wiki/Desoxyribonukleins%C3%A4ure; Abruf am 01.05.13.28.13.

[2] Nuno Roma. Design of efficient co-processing structures for dna sequence alignment. Website, 2010. http://sips.inesc-id.pt/~nfvr/msc_theses/msc09b/; Abruf am 28.07.13.

[3] Apple Inc. Opencl programming guide for mac. Website, 2013. https://developer.apple.com; Abruf am 01.05.13.