Übungen zur Vorlesung Betriebssysteme und Grundlagen der Technischen Informatik
Projekt 3c
Paralleler Vergleich von Genomsequenzen

(Version 1.1)

1 Einführung

In diesem Projekt sollen sie einen zeitaufwendigen Vergleich von Genomsequenzen mit Hilfe von Threads parallelisieren, um so die Ausführungszeit auf einer Multi-Core-Architektur zu reduzieren.

Durch dieses Projekt sollen Sie das Programmieren mit Threads, speziell mit der POSIX-Thread-Bibliothek `libpthread`, erlernen. Machen Sie sich zur Vorbereitung anhand des Vorlesungsskripts und der Man-Pages mit den grundlegenden Funktion der POSIX-Thread-Bibliothek vertraut, z.B. `pthread_create()` für das Erzeugen von Threads, `pthread_join()` für die Freigabe von Threadressourcen und `pthread_detach()` für das Abkommandieren eines Threads (selbständige Ressourcenfreigabe nach Ende des Threads).

Desweiteren sollen Sie in diesem Projekt den Umgang mit Synchronisationsmechanismen erlernen. Zur Vorbereitung machen Sie sich bitte mit den Funktionen der `libpthread` zum Erwerb und zur Freigabe von Mutex-Locks (`pthread_mutex_lock()`, `pthread_mutex_unlock()`) und zum Warten und Signalisieren bei Bedingungsvariablen (`pthread_cond_wait()`, `pthread_cond_signal()`) vertraut.

2 Leistungsumfang des Programms

2.1 Einlesen von Genomsequenzen

Als Beispiele für zu vergleichende Genomsequenzen stehen Ihnen zwei Abschnitte aus dem menschlichen X- und Y-Chromosom zur Verfügung, die homologe Gene enthalten. Es handelt sich dabei um die Datei `sox3_region.fa`, eine Region des X-Chromosoms, die auch das Gen SOX3 enthält, und um die Datei `sry.fa`, die das zu SOX3 homologe Gen SRY auf dem Y-Chromosom enthält. Diese Dateien finden Sie im Verzeichnis `/vol/lehre/GTechInf`. Die erste Zeile beider Dateien kann ignoriert werden. Es folgen Zeilen mit ASCII-Zeichen für die Nukleotidbasen. Schreiben Sie zunächst eine Funktion, mit der eine solche Datei in den Speicher einlesen werden kann. Diese Funktion soll effizient implementiert werden. Testen Sie dazu, ob Ihre Funktion schnell genug ist, um auch die vollständigen Genomsequenzen des X- oder Y-Chromosoms (`chrX.fa`, `chrY.fa`) einzulesen (ebenfalls unter `/vol/lehre/GTechInf`, bitte diese Riesendateien nicht ins Home-Verzeichnis kopieren!). Hinweis: Das zeilenweise Einlesen (z.B. mit `fgets()`) ist in der Regel zu langsam. Nutzen Sie statt dessen z.B. `fread()`, um die Datei in einem Schritt einzulesen, und eliminieren Sie anschließend die Zeilenumbrüche. Machen Sie sich dazu mit der Funktion `memmove()` und mit Zeigerarithmethik vertraut.

Die Dateien enthalten Großbuchstaben (C, G, T, A) für die Nukleotidbasen sowie Kleinbuchstaben (c, g, t, a), welche Repeats kennzeichnen; desweiteren kann der Buchstabe N für “any” auftauchen. Schreiben Sie nun eine Funktion, welche die Repeats und N aus der Genomsequenz eliminiert. Hinweis: Schauen

Sie sich die Man-Page der Funktionen `strspn()` und `strcspn()` an. Beachten Sie weiterhin, dass die Funktion `strcpy()` keine Überlappungen von Quell- und Zielbereich zulässt; verwenden Sie ggf. statt dessen `memmove()`. Testen Sie Ihre Funktion auch an den vollständigen Sequenzen aus den Dateien `chrX.fa` und `chrY.fa`.

2.2 Vergleich von Genomsequenzen

Beim Vergleich der Genomsequenzen sollen jeweils *alle* Segmente einer bestimmten Länge aus der einen mit *allen* Segmenten einer bestimmten Länge aus der anderen Sequenz verglichen werden. Der Vergleich zweier Segmente erfolgt dabei mit der Smith-Waterman-Distanz (siehe z.B. Wikipedia). Diese ist wie folgt definiert: Hat Sequenz 1 m Elemente und Sequenz 2 n Elemente, so wird eine Matrix $\mathbf{H}$ mit $(m+1) \times (n+1)$ Elementen aufgebaut. Dabei gilt für die Elemente am linken und oberen Rand:

$$H(i,0) = 0, \quad 0 \leq i \leq m \tag{1}$$
$$H(0,j) = 0, \quad 0 \leq j \leq n \tag{2}$$

Die Matrix wird nun iterativ aufgebaut. Dabei spielen drei Gewichte eine Rolle: w_{match} wird verwendet, wenn zwei verglichene Nukleotidbasen übereinstimmen, w_{mismatch}, wenn sie verschieden sind; das Gewicht w_{gap} bewertet Insertions und Deletions. Wir definieren zunächst das Gewicht beim Vergleich von zwei Basen a und b:

$$w(a,b) = \begin{cases} w_{\text{match}}, & a = b \\ w_{\text{mismatch}}, & a \neq b. \end{cases} \tag{3}$$

Die restlichen Elemente von $\mathbf{H}$ mit den Indizes $1 \leq i \leq m$ und $1 \leq j \leq n$ werden iterativ nach folgender Vorschrift ermittelt:

$$H(i,j) = \max \begin{cases} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \text{Match, Mismatch} \\ H(i-1,j) + w_{\text{gap}} & \text{Deletion} \\ H(i,j-1) + w_{\text{gap}} & \text{Insertion.} \end{cases} \tag{4}$$

Die Smith-Waterman-Distanz ist nun das Maximum aller Matrixwerte. Hinweis: Überlegen Sie, ob wirklich eine Matrix aufgebaut werden muss, oder ob es eine effizientere Implementation des Abstandsmaßes gibt.

Überprüfen Sie anschließend die Korrektheit Ihrer Implementation der Smith-Waterman-Distanz anhand des Beispiels (Wikipedia): Sequenz 1 = ACACACTA, Sequenz 2 = AGCACACA. Verwenden Sie hier und im Folgenden stets $w_{\text{match}} = 2$, $w_{\text{mismatch}} = -1$, $w_{\text{gap}} = -1$. Die Matrix sollte dann folgende Elemente haben (die Sequenzen sind zur Orientierung am Rand notiert):

	-	A	C	A	C	A	C	T	A
-	0	0	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1	0	2
G	0	1	1	1	1	1	1	0	1
C	0	0	3	2	3	2	3	2	1
A	0	2	2	5	4	5	4	3	4
C	0	1	4	4	7	6	7	6	5
A	0	2	3	6	6	9	8	7	8
C	0	1	4	5	8	8	11	10	9
A	0	2	3	6	7	10	10	10	12

Die Smith-Waterman-Distanz, also das maximale Element in der Matrix, ist in diesem Beispiel 12.

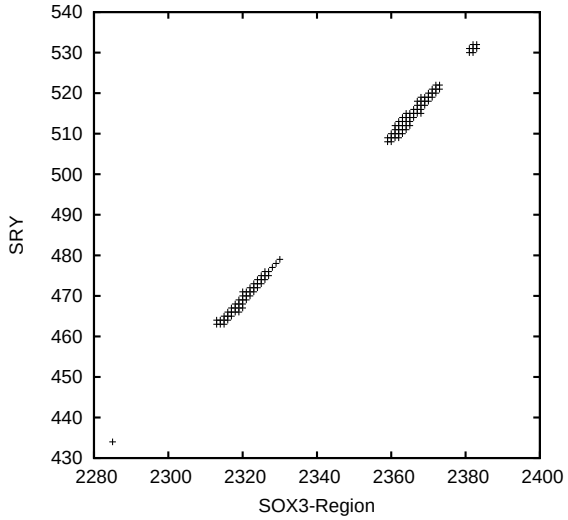
2.3 Parallelisierung des Genomvergleichs

Die Sequenzen der SOX3-Region und des SRY-Gens sollen nun eingelesen und verglichen werden. Dazu wird jedes Segment mit 50 aufeinanderfolgenden Basen aus der einen mit jedem Segment mit 50 aufeinanderfolgenden Basen aus der anderen Sequenz verglichen (d.h. $m = n = 50$). Liefert der Vergleich eine Smith-Waterman-Distanz von mindestens 70, so sollen die Anfangskoordinaten der Segmente und die Distanz in einer Liste gespeichert werden.

Die Parallelisierung mit mehreren Threads soll so erfolgen, dass die längere Sequenz (z.B. kann das immer die erste sein) in mehrere Regionen unterteilt wird. Jede dieser Regionen der ersten Sequenz soll von einem eigenen Thread mit der kompletten zweiten Sequenz verglichen werden (alle Segmente gegen alle Segmente). (Wie können Sie sicherstellen, dass an der Grenze zwischen den Regionen in der ersten Sequenz keine Vergleiche ausgelassen oder doppelt durchgeführt werden? Wie können Sie beim Aufteilen in Regionen sicherstellen, dass alle Basen verglichen werden?)

Entwerfen Sie Ihr Programm so, dass als Kommandozeilenparameter die Anzahl der zu startenden Threads übergeben werden kann. Implementieren Sie den Segment-Vergleich in einer Thread-Funktion, die beim Start des Threads mit `pthread_create()` aufgerufen wird. Entwerfen Sie eine Datenstruktur, die via Zeiger an jede Threadfunktion übergeben werden kann. Diese sollte alle Daten enthalten, die vom Thread benötigt oder erzeugt werden. Überlegen Sie, wie Sie die Ergebnisliste von den Threads erzeugen lassen können, ohne dass sich diese dabei wechselseitig beeinflussen (sowohl in fehlerhafter Weise als auch einer Weise, die die Ausführungsgeschwindigkeit beeinträchtigt).

Geben Sie die Ergebnisliste in eine Datei aus. Visualisieren Sie diese Daten, z.B. mit `gnuplot`. Dies sollte folgende Matches ergeben:

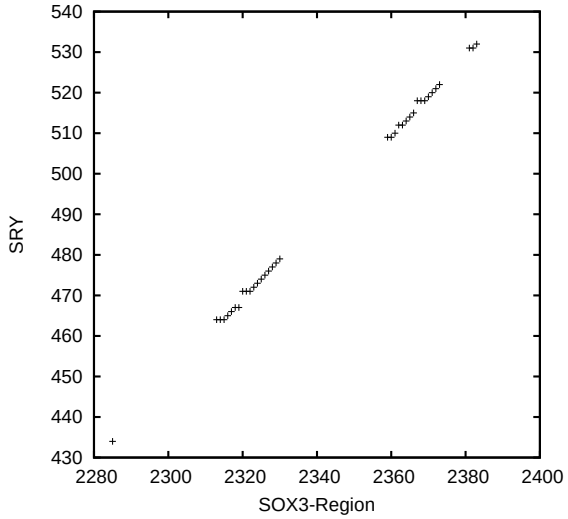


Vergleichen Sie die Ausführungszeit Ihres Programms mit unterschiedlicher Anzahl von Threads. Sie können dazu das Kommando `time` zum Start Ihres Programms verwenden. Was stellen Sie fest?

Hinweise: Endet die Funktion `main()`, so enden auch alle Threads. Verwenden Sie deshalb `pthread_join()`, um auf das Ende der Threads zu warten. Beachten Sie, dass auf mit `pthread_detach()` abkommandierte Threads nicht mit `pthread_join()` gewartet werden darf.

2.4 Nachbearbeitung der Daten

Erweitern Sie nun Ihr Programm um eine Nachbearbeitung der Daten. Dazu soll der Hauptthread des Programms (der die Funktion `main()` ausführt) von jedem der Vergleicherthreads mittels einer Bedingungsvariable informiert werden, sobald der Vergleich *eines* Segments in der vom Thread bearbeiteten Region aus Sequenz 1 mit *allen* Segmenten aus Sequenz 2 abgeschlossen ist. Die Vergleichsergebnisse (Liste der überschwelligen Matches) sollen dann sofort vom Vergleicherthread an den Hauptthread übermittelt werden. Dieser soll die Daten nachbearbeiten, während der Vergleicherthread mit seiner Arbeit fortfährt. Bei der Nachbearbeitung der Daten soll aus allen passenden Segmenten (mit überschwelliger Smith-Waterman-Distanz) aus der zweiten Sequenz genau eines mit maximaler Smith-Waterman-Distanz isoliert werden. Nachdem alle Threads abgeschlossen sind, sollen die nachbearbeiteten Daten in eine Datei ausgegeben und visualisiert werden. Dies sollte in etwa folgendes Ergebnis hervorbringen:



Verwenden Sie zur Kommunikation zwischen Vergleicherthreads und Hauptthread eine Bedingungsvariable in Kombination mit einem Mutex-Lock. Über diese Bedingungsvariable soll sowohl signalisiert werden, dass neue Daten zur Nachbearbeitung vorliegen, als auch, dass der Thread beendet wurde. Überlegen Sie, wie die Übergabe der Daten erfolgen kann, so dass der Vergleicherthread sofort seine Arbeit fortsetzen kann und nicht auf die Nachbearbeitung warten muss. Beachten Sie dabei auch, dass der Hauptthread unter Umständen erst nach langer Zeit vom Scheduler wieder auf die CPU gelangen kann. Denken Sie an den Schutz gemeinsam verwendeter Daten über das mit der Bedingungsvariable assoziierte Mutex-Lock. Beachten Sie, dass ein Signal an eine Bedingungsvariable wirkungslos ist, wenn niemand auf das Signal wartet (d.h., das Signal geht verloren). Beachten Sie auch, dass Pthread-Bedingungsvariablen auch aus `pthread_cond_wait()` aufwachen können, ohne dass dies mit `pthread_cond_signal()` signalisiert wurde. Testen Sie deshalb die Wartebedingung immer in einer Schleife.

Vergleichen Sie die Laufzeit Ihres Programms (Kommando `time`) mit Ihrer vorherigen Version ohne Nachbearbeitung (es empfiehlt sich die Verwendung von Präprozessor-Anweisungen, um die zwei Versionen des Programms in einer einzigen Quelldatei realisieren zu können). Verbessern Sie Ihr Programm, falls eine deutliche Verschlechterung der Laufzeiten zu beobachten ist.

Allgemeiner Hinweis: Verwenden Sie die Option -pthread bei gcc. Dies setzt notwendige Präprozessor-Definitionen und sorgt dafür, dass mit der Thread-Bibliothek gelinkt wird.