Prof. Dr. Knut Reinert
René Rahn
Jakob Schulze

Institut für Informatik
AG Algorithmische Bioinformatik

Algorithmen und Datenstrukturen in der Bioinformatik
4. Programmieraufgabe WS 14

Abgabe Montag, 15.12., 15:00 Uhr per SVN

Parallelisierte Wordliste

In der Vorlesung wurde der Algorithmus BLAST zum Auffinden lokaler Ähnlichkeiten zwischen einer Sequenz und einer Datenbank besprochen. Implementieren Sie das Generieren der Nachbarschaft eines Wortes der Länge w für alle Wörter einer gegebenen Sequenz. Im folgenden bekommt Ihr Programm (aufgabe4.cpp) folgende Kommandozeilenargumente mit übergeben.

a) `''SEQ''` - Eine Aminosäurensequenz.

b) `<SCORE_MATRIX>` - Eine Datei mit der Scoring-Matrix, z.B. Blosum62. (Hinweis: Funktionen zum Lesen und Arbeiten mit den Matritzen werden gestellt.)

c) `<WORD_SIZE>` - Die Länge der zu generierenden Wörter.

d) `<THRESHOLD>` - Der Grenzwert für welches ein Nachbarwort mit in die Liste übernommen wird.

e) `<THREADS>` - Die Anzahl der Threads für die parallele Verarbeitung. Default 1.

Im SVN im Ordner `material/material_a4/` finden Sie verschiedene Scoring-Matritzen sowie eine Headerdatei "a4_util.h". Sie können diese Headerdatei mittels `#include ''../material/material_a4/a4_util.h''` in Ihrem Programm einbinden. In dieser Headerdatei stehen Funktionen bereit um die Scoring-Matritzen einzulesen und um die Scoring-Matrix zu benutzen.

Schreiben Sie eine Funktion mit dem Namen `generateNeighborhood`. Diese Funktion bekommt folgende Parameter übergeben.

- `const std::string & seq`: Die Sequenz die dem Programm am Anfang übergeben wurde.
- `const ScoreMatrix & matrix`: Die Scoring-Matrix für das Berechnen der Nachbarschaft.
- `unsigned wordSize`: Die zu verwendende Wortgröße.
- `int threshold`: Der minimale Score den ein Wort haben muss um zur Nachbarschaft zugehören.
- `unsigned threads`: Die Anzahl der Threads die verwendet werden sollen.

Die Funktion iteriert über jedes w-Wort der gegebenen Sequenz. Für jedes w-Wort wird die Nachbarschaft gegeben der Scoring-Matrix und des Thresholds berechnet und anschließend in dem Format:

```
word_1: (n1, s1) (n2, s2) (n3, s3) ...
word_2: (n1, s1) (n2, s2) (n3, s3) ...
...
```

ausgegeben. Dabei wird für jedes Wort der Sequenz in einer neuen Zeile das Wort selber und dann durch ein Doppelpunkt getrennt die Liste der Nachbarn zu diesem Wort ausgegeben. Die Nachbarn werden als Tupel dargestellt, wobei das erste Element das Nachbarwort ist und das zweite Element der entsprechende Score für diesen Nachbarn ist. Achten Sie auf das Format `KLAMMER-AUF WORT KOMMA SCORE KLAMMER-ZU`! Achten Sie darauf, dass die ausgegebenen Wörter der Reihenfolge in der Sequenz entsprechen. Also in der gleichen Reihenfolge wie die sequentielle Abarbeitung ergeben würde.

Messen Sie die Zeit die Ihr Programm benötigt um die Liste der Nachbarn zu generieren. Geben Sie die Zeit am Ende mit aus (Siehe Beispiel! Auf das Format achten!!!!)

Beispiel

```
talisker$ ./aufgabe4 "AAHILNMY" blosum62 3 13 4
AAH: (AAH, 16)
AHI: (AHI, 16) (AHL, 14) (AHV, 15)
HIL: (HII, 14) (HIL, 16) (HIM, 14) (HLL, 14) (HVL, 15)
ILN: (ILN, 14)
LNM: (LNM, 15)
NMY: (NIY, 14) (NLY, 15) (NMF, 14) (NMY, 18) (NVY, 14)
time: 0.13s
```

Praktikumshinweise Beachten Sie die Hinweise unter `https://www.mi.fu-berlin.de/w/ABI/AlDaBiWS14Praktikum`.