

Übungsblatt 02: Speicherhierarchie richtig nutzen

7. November 2019

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 15.11.2018, 11:59 Uhr**.
- Abgabe an hannah.muelbaier@gmail.com UND carla.moelbert@gmx.de

Aufgabe 2.1: Matrizenmultiplikation (10 Punkte)

Eine Matrizenmultiplikation ist ein gutes Beispiel dafür, wie man die Speicherhierarchie nicht oder auch optimal nutzen kann.

Implementiere eine Matrizenmultiplikation in C/C++ (d.h. lauffähig, kein Pseudocode!):

- Der Einfachheit halber haben die zu multiplizierenden Matrizen die gleichen Dimensionen und sind quadratisch: also z.B. 8x8 Felder groß
- Beachte, dass bei den in Aufgabe 2.2 geforderten Dimensionen ein statisches Array nicht funktioniert, d.h. die Arrays müssen dynamisch sein (Stichworte: Heap, Stack, Speicherverwaltung)
- Die Abbildung der zweidimensionalen Matrizen erfolgt auf eindimensionale Arrays
- Die Quellmatrizen sollen mit zufälligen Integer-Werten gefüllt, die Zielmatrize mit 0 initialisiert
- Implementiere drei Varianten, welche die Berechnung in verschiedenen Reihenfolgen abarbeiten:

<p>Variante 1 (ijk) – innere Schleife</p> $\begin{pmatrix} (i,*) \\ \boxed{} \cdots \boxed{} \\ A \end{pmatrix} \times \begin{pmatrix} (*,j) \\ \boxed{} \\ \vdots \\ \boxed{} \\ B \end{pmatrix} = \begin{pmatrix} \boxed{} (i,j) \\ C \end{pmatrix}$ <p>Variante 2 (kij) – innere Schleife</p> $\begin{pmatrix} \boxed{} (i,k) \\ A \end{pmatrix} \times \begin{pmatrix} (k,*) \\ \boxed{} \cdots \boxed{} \\ B \end{pmatrix} = \begin{pmatrix} (i,*) \\ \boxed{} \cdots \boxed{} \\ C \end{pmatrix}$ <p>Variante 3 (jki) – innere Schleife</p> $\begin{pmatrix} (*,k) \\ \boxed{} \\ \vdots \\ \boxed{} \\ A \end{pmatrix} \times \begin{pmatrix} \boxed{} (k,j) \\ B \end{pmatrix} = \begin{pmatrix} (*,j) \\ \boxed{} \\ \vdots \\ \boxed{} \\ C \end{pmatrix}$

Aufgabe 2.2: Auswertung (10 Punkte)

Führe dein Programm mehrfach mit den folgenden Matrixdimensionen aus: 64, 128, 256, 512, 1024, 2048.

Achtung: die Ausführung für die Größen 1024 und 2048 kann mehrere Minuten dauern!

Notiere die aus den mehrfachen Durchläufen berechneten mittleren Laufzeiten je Dimension:

- (a) für die Variante 1 (Zeile x Spalte = Zielfeld)
- (b) für die Variante 2 (Quellfeld x Zeile = Zielzeile)
- (c) für die Variante 3 (Spalte x Quellfeld = Zielspalte)

Beantworte die folgenden Fragen:

- (a) Welche Variante ist die schnellste?
- (b) Welche Variante ist die langsamste?
- (c) Überlege dir dazu, wie die Daten im Arbeitsspeicher angeordnet sind und wie sie von dort zur CPU transportiert werden. Beschreibe dann kurz, wie die unterschiedlichen Laufzeiten zustande kommen.

Codeschnipsel

```
1 #include <stdlib.h>
2 #include <sys/time.h>
3 #include <time.h>
4 #include <stdio.h>
5 #include <unistd.h>
6
7 int main (int argc, const char* argv[]) {
8
9     // teste, ob ein Argument uebergeben wird
10    // falls nicht, brich das Programm ab
11    if (argc < 2) {
12        return 1;
13    }
14
15    // die Dimension der Matrizen entspricht dem Argument
16    int dim = atoi(argv[1]);
17
18    printf("Integergroesse: %lu\n", sizeof(int));
19    printf("Matrixgroesse: %d x %d\n", dim, dim);
20    printf("Die Matrizen belegen jeweils %lu bytes an Speicher.\n",
21        dim * dim * sizeof(int));
22    printf("Insgesamt also %lu bytes.\n", 3 * dim * dim * sizeof(int));
23
24    // AUFGABE: "erzeuge" hier die Matrizen
25
26    // Messinstrumente
27    struct timeval start, end;
28    long mtime, seconds, useconds;
29
30    // Initialisiere Random Number Generator
31    srand((unsigned) time(NULL));
32
33    // AUFGABE: fuelle hier die Matrizen
34
35    // starte die Zeitnahme
36    gettimeofday(&start, NULL);
37
38    // AUFGABE: multipliziere hier die Matrizen
39    /* Hinweis: Kleine Matrizen auszugeben ist zwar nuetzlich,
40     um die Berechnung zu ueberpruefen, aber es kostet viel Zeit.
41     Darum hat es nichts im entgueltigen Code zu suchen, kann aber
42     bei der fehlerfreien Implementierung helfen.
43     */
44
45    // beende die Zeitnahme und gib die Dauer der Berechnung aus
46    gettimeofday(&end, NULL);
47    seconds = end.tv_sec - start.tv_sec;
48    useconds = end.tv_usec - start.tv_usec;
49    mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
50    printf("Die Ausfuehrung dauerte %ld ms.\n", mtime);
51
52    // AUFGABE: Raeume den Speicher auf
53
54    // beende das Programm
55    return 0;
56 }
```

Listing 1: Codeschnipsel zur Matrizenmultiplikation