Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Christian Engwer, MSc. Marcel Koch

9.11.2017

Übung zur Vorlesung

Wissenschaftliches Rechnen

WS 2017/18 — Blatt 5

Abgabe: 16.11.2017, 10:00 Uhr, Briefkasten 111
Code zusätzlich per e-mail an `marcel.koch@uni-muenster.de`

Aufgabe 1 (Loop Order) (2 Punkte)

Ein Matrix-Vektor Produkt lässt sich auf folgende zwei Arten realisieren:

```
for(int i = 0; i < 256; ++i){
  for(int j = 0; j < 640; ++j){
    y[i] += A[i][j] * x[j];
  }
}
```

(i) Innerste Schleife über die Spalten

```
for(int j = 0; j < 640; ++j){
  for(int i = 0; j < 256; ++i){
    y[i] += A[i][j] * x[j];
  }
}
```

(ii) Innerste Schleife über die Zeilen

Gehen Sie davon aus, dass die Arrays vom Typ `double` (8 Byte) bereits sinnvoll initialisiert sind. Die Matrix `A` ist spaltenweise gespeichert, d.h. Einträge in der selben Spalte liegen im Speicher nebeneinander. Der Eintrag `A[i][j]` liegt somit an Stelle `[i+640*j]` nach dem ersten Eintrag `A[0][0]`.
Betrachten Sie einen vollassoziativen L1-Datencache eines Prozessorkerns mit einer Größe von 32kB (1 kB = 1024 Byte) und einer Cachelinegröße von 64 Byte.

(a) Von welcher Variante erwarten Sie eine bessere Performance? Begründen Sie ihre Entscheidung.

(b) Gibt es Matrizengrößen, bei denen ein Unterschied nicht mehr feststellbar ist?

Aufgabe 2 (1D-Wärmeleitungsgleichung mit Cache-Effekten) (3 Punkte)

Betrachten Sie die homogene Wärmeleitungsgleichung in 1D mit Dirichlet Randbedingungen:

$$\begin{aligned} \partial_t u - \partial_{xx} u &= 0 \quad \text{in } (0,T] \times (0,1), \\ u(t,0) &= u_a, \\ u(t,1) &= u_b. \end{aligned} \tag{1}$$

Diskretisieren Sie die Gleichung im Ort mit dem zentralen Differenzenquotienten und in der Zeit mit dem expliziten Eulerverfahren. Untersuchen Sie die Anzahl an Gleitkommaoperationen

pro Sekunde (FLOPS) die Ihr Code für verschieden feine Ortsdiskretisierungen erreicht. Stellen Sie die Ergebnisse graphisch dar und und diskutieren Sie diese. Bestimmen Sie anhand Ihrer Ergebnisse approximativ die Cachegrößen.

Aufgabe 3 (Tiling beim Matrix-Matrix Produkt) (4 Punkte)

Betrachten Sie folgende Implementierung eines Produkts zweier dicht besetzten Matrizen:

```
for(int i = 0; i < N; ++i)
  for(int j = 0; j < N; ++j)
    for(int k = 0; k < N; ++k)
      C[i][j] += A[i][k] * B[k][j];
```

Wie in Aufgabe 1 wird wieder ein Prozessorkern mit vollassoziativen L1-Datencache der Größe 32kB und einer Cachelinegröße von 64 Byte betrachtet.

(a) Bestimmen Sie die Größe `N` der Matrizen, ab der eine ineffiziente Cache Ausnutzung eintritt. Ab dieser Größe ist es sinnvoll, die Matrizen in `m`×`m` große Blöcke zu zerlegen und das Produkt als Blockmatrix Produkt umzusetzen. Wieso lässt sich dadurch ein Performancegewinn erreichen für bestimmte Blockgrößen erreichen. Für welche Blockgrößen würden Sie die meiste Verbesserung erwarten?

(b) Implementieren Sie ein geblockte und eine nicht geblockte Version und vergleichen Sie die FLOPS für unterschiedliche Matrix- und Blockgrößen.

Bemerkung 1 (Hinweise zu Aufgabe 2 und 3)

Neueste Kompiler können stärker optimieren als Ihnen vielleicht bewusst ist. Dies kann bei der Zeitmessung von kleinen Code-Aussschnitten zu unerwarteten Ergebnissen führen. Beispielsweise möchten Sie die Zeit für eine skalierte Vektoraddition messen:

```
for(int i = 0; i < MAX_ITER; ++i){
  for(int d = 0; d < DIM; ++d)
    y[i] += alpha*x[i];
}
```

Hier kann es passieren, falls Sie nicht mehr auf Werte von `y` zugreifen, der Kompiler beide Schleifen, und damit auch den zu testenden Code, entfernt. Um dies zu verhindern fügen Sie folgende Befehle ein:

```
asm volatile("" : : "i,r,m"(y) : "memory");

for(int i = 0; i < MAX_ITER; ++i){
  for(int d = 0; d < DIM; ++d)
    y[i] += alpha*x[i];

  asm volatile("" : : : "memory");
}
```

Der Kompiler ist dann gezwungen beide Schleifen zu erhalten und somit ist eine sinnvolle Zeitmessung möglich.