Übungen zur Vorlesung

Paralleles Höchstleistungsrechnen

Dr. S. Lang, J. Pods

Abgabe: 13. November 2012, 12 Uhr, per E-mail an `jurgis.pods@iwr.uni-heidelberg.de`

Übung 3 C++-Einführung: Debugging (5 Punkte)

Das folgende Programm soll alle natürlichen Zahlen von gegebenen $a \in \mathbb{R}$ bis $b \in \mathbb{R}$ summieren:

```
1 #include <iostream>
2
3 // sums all natural numbers in [a, b]
4 int sum(int a , int b)
5 {
6   int result;
7   for (int i=a; i<=b ; i++)
8   {
9     int result = result + i;
10  }
11
12  return 0 ;
13 }
14
15 int main()
16 {
17  std::cout << sum(1, 10) << std::endl;
18  return 0 ;
19 }
```

Obwohl das Programm syntaktisch korrekt ist, berechnet es ein falsches Ergebnis. Finden Sie die Fehler und korrigieren Sie diese, ohne das Programm logisch zu verändern. Tipp: Falls Sie das Programm am Rechner in einer Datei `debug.cc` ausprobieren wollen, gibt Ihnen der C++-Compiler mit der Option `-Wall` Hinweise zu fehlerhaftem Code. In der Kommandozeile übersetzen: `g++ -Wall debug.cc`.

Übung 4 Messen von MFLOPS (15 Punkte)

In dieser Übung wollen wir für zwei numerische Anwendungen ausprobieren, wieviele Rechenoperationen pro Sekunde heute möglich sind. Dazu wollen wir folgende mathematische Operationen implementieren:

1. *Matrix-Multiplikation.*
   Es seien zwei Matrizen $A, B \in \mathbb{R}^{n\times n}$ gegeben. Dann ist das Matrixprodukt $C = AB$ wiederum eine Matrix $C \in \mathbb{R}^{n\times n}$ mit den Einträgen:
   $$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

2. *Gauß-Seidel 2d.*
   Es sei ein Gebiet in d Dimensionen definiert über
   $$\Omega_n^d = \left\{ (i_0, \ldots, i_{d-1}) \in \mathbb{Z}^d \mid \forall\, 0 \le k < d : 0 \le i_k < n \right\}.$$
   In $2D$ wäre dies zum Beispiel ein Netz mit n^2 Punkten. Wir wollen ein Gitter mit äquidistanten Punkten wählen, d. h. $\Omega = [0, n-1]^2$. Auf diesem Gitter sei eine Gitterfunktionen $u^m : \Omega_n^2 \to \mathbb{R}$ gegeben. Für diese definiert die Iterationsvorschrift
   $$u^{m+1}(i,j) = \frac{1}{4}\Big\{ u^{m+1}(i-1,j) + u^{m+1}(i,j-1) + u^m(i,j+1) + u^m(i+1,j) \Big\} \quad (i,j) \in [0, n-1]^2$$
   die sogenannte Gauss-Seidel-Iteration.

Teilaufgabe (a) (5 Punkte)

Implementieren Sie die Matrix-Multiplikation in der Programmiersprache C/C++ und verwenden Sie für die Matrizen eine beliebige Datenstruktur ihrer Wahl (z. B. eindimensionale oder zweidimensionale Felder oder `std::vector`). Bestimmen Sie die Anzahl der Fließkomma-Operationen und ermitteln Sie daraus und aus der gemessenen Laufzeit die Geschwindigkeit des Programmes in „Millionen Fließkomma-Operationen pro Sekunde“.

Zur Zeitmessung können Sie die Funktionen aus `timer.h` verwenden. Den Header finden Sie auf der Vorlesungs-Homepage, Hinweise zur Verwendung am Ende des Übungsblattes. Achten Sie darauf, daß Sie die Problemgröße n so groß wählen, daß die Zeitmessung nicht durch Zeitmessfehler verfälscht wird, im Pool etwa $n \geq 1000$. Initialisieren Sie die Felder mit sinnvollen Daten (nicht 0.0), z. B. $u(i, j) = i + j$.

Übersetzen Sie das Programm mit maximaler Optimierungsstufe. Für den `GNU C/C++` Compiler ist etwa `-O3 -funroll-loops` empfehlenswert.

Stellen Sie alle Ergebnisse in graphischer Form, MFLOPS über Problemgröße n dar. Diskutieren Sie die Form der Kurven, insbesondere warum die MFLOP-Rate wann absinkt. Zur graphischen Darstellung eignet sich das auch im Pool installierte Programm `gnuplot`. Eine kurze Einführung findet sich auf der Homepage, gut ist außerdem die Einführung von D. Völker der FU Berlin:
http://userpage.fu-berlin.de/~voelker/gnuplotkurs/gnuplotkurs.html

Teilaufgabe (b) (5 Punkte)

Wiederholen Sie die Untersuchungen aus Teilaufgabe (b) für das Gauss-Seidel-Verfahren.

Teilaufgabe (c) (5 Punkte)

Führen Sie für die Matrix-Multiplikation eine bessere Cache-Nutzung durch Kachelung ein, wie in der Vorlesung besprochen, und ermitteln Sie die Beschleunigung für verschiedene Kachelgrößen.

Hinweise zur Zeitmessung

Die verschiedenen Zeiten

Bei der Zeitmessung am Computer ergibt sich das Problem, dass die Zeit, die ein Programm benötigt, von der Auslastung des Systems abhängt. Sind viele Prozesse tätig, bekommt der einzelne nur wenig Zeit und läuft dementsprechend lange. Die Prozessorzeit gibt hingegen an, wieviele Prozessorsekunden das Programm verbraucht hat. Die Uhr tickt, solange das Pogramm läuft, wenn das Betriebssystem das Programm warten läßt, steht sie.

timer.h

In der Headerdatei `timer.h` sind einige Hilfsfunktionen implementiert, die die verbrauchte Prozessorzeit auslesen. Sie stellt drei Befehle zur Verfügung:

- `void reset_timer(struct timeval* timer)`: Zähler zurücksetzen/initialisieren.
- `double get_timer(struct timeval timer)`: verbrauchte Sekunden auslesen.
- `void print_timer(struct timeval timer)`: verbrauchte Sekunden ausgeben.

Beispiel

```
1 #include "timer.h"       // Headerfile zur Zeimessung
2
3 int main()
4 {
5   struct timeval timer; // Variable zur Zeitmessung
6   reset_timer(&timer);  // Zaehler zuruecksetzen/
    initialisieren
```

```
7     ...                       // Was tun und Zeit verbrauchen
8     print_timer(timer);       // Zaehler ausgeben
9  }
```

Mehr über die Interna der Zeitmessung kann in der manpage zu `getrusage (2)` weitere Informationen nachgelesen werden.