Übungen zur Vorlesung

Wissenschaftliches Rechnen – Paralleles Höchstleistungsrechnen

Prof. Dr. C. Engwer, S. Westerheide
http://wwwmath.uni-muenster.de/num/Vorlesungen/WissenschaftlichesRechnen_SS11/

Abgabe 20. 6. 2011. Abg. der Programmieraufgaben per Email an sebastian.westerheide@uni-muenster.de, schriftliche Abgabe **in Zimmer 120.019**.

- Alle Programmierübungen müssen per Email und in ausgedruckter Form abgegeben werden.
- Achten sie darauf, ihr Programm ordentlich zu formatieren und gut zu kommentieren.

Übung 1 Skalierter Speedup

Das Skalarprodukt $s = x \cdot y = \sum_{i=0}^{N-1} x_i y_i$ zweier Vektoren $x, y \in \mathbb{R}^N$ soll auf einem Hypercube mit P Prozessoren parallel berechnet werden. Zur Vereinfachung sei $P = 2^d, d \in \mathbb{N}_0$ und N durch P teilbar.

Wir gehen folgendermaßen vor:

- Jeder Prozessor p hat N/P Komponenten von x und y mit den Indizes $I_p \subset \{0, \ldots, N-1\}$ im lokalen Speicher und berechnet im ersten Schritt die Teilsumme $s_p = \sum_{i \in I_p} x_i y_i$.
- Die P Zwischenergebnisse werden dann im Baum addiert (vgl. Figure 1).

```
parallel message-passing-scalar-product
{
    const int d, P= 2^d, N;

    process Π [int p ∈ {0,...,P-1}]
    {
        double x[N/P], y[N/P];        // Lokaler Ausschnitt der Vektoren
        int i, r, m;
        double s = 0, ss;

        for (i = 0; i < N/P; i++) s = s + x[i] * y[i];

        for (i = 0; i < d; i++)       // Summation im Baum (d Schritte)
        {
            r = p & [~ ( Σ_{k=0}^{i} 2^k ) ];   // Bit 0 bis Bit i auf 0 setzen
            m = r | 2^i;              // Bit 0 bis Bit i-1 auf 0 setzen, Bit i auf 1
            if (p == m)
                send(Π_r,s);
            if (p == r)
            {
                receive(Π_m,ss);
                s = s + ss;
            }
        }
    }
}
```

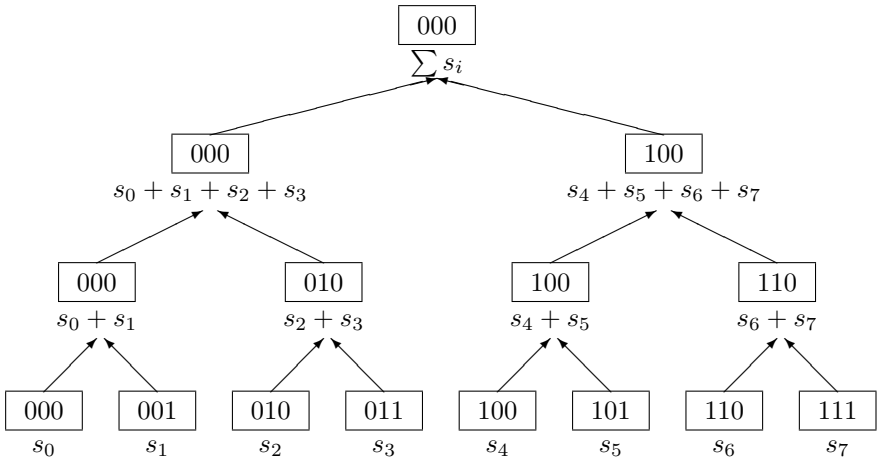

Figure 1: Parallele Summation im Baum

Aufgabenteil a)

Untersuchen Sie die Leistungsfähigkeit dieses parallelen Systems. Berechnen Sie dazu in Abhängigkeit der Zeit t_f für eine arithmetische Operation und der Zeit t_m für den Nachrichtenaustausch

- die Laufzeit $T_{\text{best}}(N)$ des besten sequentiellen Algorithmus' auf einem Knoten des Parallelrechners,
- die Laufzeit $T_P(N, P)$ des parallelen Algorithmus',
- den Speedup $S(N, P)$.

Aufgabenteil b)

Untersuchen Sie nun, wie sich der Speedup $S(N, P)$ in Abhängigkeit von P verhält. Skalieren Sie ihn dazu durch die Berechnung geeigneter Problemgrößen

- $N = N_A$,
- $N = N_G(P)$,
- $N = N_M(P)$,
- $N = N_I(P)$

so, dass sich eine feste sequentielle Ausführungszeit $\big(T_{\text{best}}(N) = T_{\text{fix}}\big)$, eine feste parallele Ausführungszeit $\big(T_P(N, P) = T_{\text{fix}}\big)$, ein fester Speicherbedarf pro Prozessor $\big(M(N) = M_0 P\big)$, oder eine konstante Effizienz $\big(E(N, P) = E_0\big)$ ergibt. Ermitteln Sie jeweils den resultierenden Speedup

- $S_A(P) = S\big(N_A, P\big)$,
- $S_G(P) = S\big(N_G(P), P\big)$,
- $S_M(P) = S\big(N_M(P), P\big)$,
- $S_I(P) = S\big(N_I(P), P\big)$.

Machen Sie sich jeweils Gedanken zur Größenordnung des Speedups (bezüglich P) und vergleichen Sie die Größenordnungen für die verschiedenen Skalierungen.

8 Punkte

Übung 2 N-Körper-Problem mit PThreads

a) Parallelisieren Sie das N-Körper-Problem mit `PThreads`. Laden Sie dazu die Datei `nbody_seq.cc` von der Vorlesungsseite, kopieren Sie es nach `nbody_pthread.cc` und parallelisieren Sie (nur) die Funktion `acceleration()`, welche die Hauptlast der Berechnungen trägt (sie hat die Komplexität $O(N^2)$, alles andere hingegen $O(N)$). Das seq. Programm implementiert die Berechnung der Beschleunigung mit Hilfe von Tiling und Ausnutzung der Symmetrie.

- Implementieren Sie `acceleration()` als daten-paralleles Programm.
- Welche parallelen Datenaufteilung wählen Sie und warum?
- Welche Bereichen müssen abgesichert werden?
- Sollte eventuell der Algorithmus irgendwo modifiziert werden?

b) Vergleichen Sie die Zeiten des parallelen Programms mit der sequentiellen Variante und stellen Sie die Messungen grafisch dar. Denken Sie daran, dass immer mit dem *besten* sequentiellen Programm verglichen werden soll.

- Messen Sie den Speedup bei fester sequentieller Ausführungszeit, d.h. für eine feste Partikelanzahl.
- Messen Sie den Speedup bei festem Speicherbedarf pro Prozessor, indem Sie die Partikelanzahl geeignet in Abhängigkeit von der Anzahl der verwendeten Prozessoren wählen.

c) Verifizieren Sie ferner experimentell die Komplexität des Algorithmus', indem Sie für eine feste Anzahl von Prozessoren die Partikelanzahl variieren.

Auf der Vorlesungs-Homepage finden Sie die benötigten Programmteile sowie ein Beispiel Video der Rechnung (`collision.avi`), dieses wurde mit folgenden Programm-Parametern berechnet:

```
./nbody_pthread 5000 3000000 15 200 4
```

Das ist eine Rechnung mit 5000 Partikeln, über einen Zeitraum von 3000000 Jahren. Das Programm rechnet mit 4 Threads, jeder zeitschritt ist 200 Jahre lang und jeder 15. Zeitschritt wird herausgeschrieben.

Die Anfangskonfiguration beschreibt in diesem Beispiel zwei Galaxien, welche im Laufe der Simulation kollidieren.

Hinweis 1: Thread-Funktionen können immer nur einen Pointer als Parameter bekommen. Verwenden Sie einen zusammengesetzte Datentyp `struct acceleration_data` um alle Parameter und benötigten Daten an acceleration zu übergeben.

Hinweis 2: Denken Sie daran beim Kompilieren die Parameter zur Optimierung mit anzugeben. Im Computerpool können Sie beispielweise die folgenden Parameter verwenden:

```
-O3 -ffast-math -funroll-loops -fexpensive-optimizations -march=core2
```

Zusätzlich benötigen Sie noch die Parameter für die Bibliotheken

```
-lpthread -lgomp
```

Eine Übersicht über die möglichen Compilerparameter erhalten Sie mit dem Kommando `man gcc`.

Hinweis 3: Um die parallele Variante auf Richtigkeit zu prüfen, können Sie die generierten VTK-Dateien vergleichen. Die Simulations-Ergebnisse können sich aber in einigen Nachkommastellen unterscheiden. Das auf der Vorlesungsseite bereitgestellte Skript `fuzzy_diff` kann zwei VTK-Dateien vergleichen bezüglich einer vorgegebenen Toleranz:

```
./fuzzy_diff sequential.vtk parallel.vtk 1e-10
```

12 Punkte