Übungen zur Vorlesung

Paralleles Höchstleistungsrechnen

Dr. S. Lang, D. Popović

Abgabe: 15. Dezember 2009 in der Übung

Übung 17 Isoeffizienz-Analyse des Merge-Sort (10 Punkte)

Ein möglicher Algorithmus, um eine ungeordnete Liste zu sortieren, ist der *Mergesort*-Algortihmus. Für eine n-elementige Liste hat er hat die Komplexität $O(n\,logn)$ und folgendes Schema in Pseudocode-Notation:

```
List Mergesort(List mylist)
{
  if (mylist.size() == 1) {
    return mylist;
  }
  else {
    part1 ← Mergesort(mylist[0..n/2];
    part2 ← Mergesort(mylist[n/2+1..n]);
    return Merge(part1, part2);
  }
}
```

Der Algorithmus teilt die Liste also rekursiv auf, bis nur noch ein Element sortiert werden muss. Anschliessend wird in umgekehrter Reihenfolge durch die Funktion $Merge()$ wiedervereint.

Entwickeln Sie eine parallele Version des Algorithmus unter folgenden Annahmen:

- Ein Prozess enthält zu Beginn die unsortierte Liste.
- Am Ende hält dieser Prozess wieder die nun sortierte Liste.
- Für die Anzahl der Prozessoren gilt: $p = 2^i, i = 1, 2, ...$

Beantworten Sie dann folgende Fragen:

(a) Wie groß ist die reine Kommunikationszeit in Abhängigkeit der Problemgröße n für einen Schritt und insgesamt?
(b) Wie groß ist „Rechenzeit“ (Zeit für Vergleiche) in Abhängigkeit der Problemgröße n für einen Schritt und insgesamt?
(c) Wie sind also die gesamte parallele Laufzeit und der Speed-Up?
(d) Können Sie eine Isoeffizienz-Funktion angeben?

Implementieren Sie nun eine parallele Variante des Algorithmus und testen Sie ihn für verschiedene n und p. Können Sie die vorhergesagten Ergebnisse reproduzieren?

Übung 18 Isoeffizienz-Analyse des Jacobi-Verfahrens (10 Punkte)

Auf den letzten Übungszetteln haben Sie das Jacobi-Verfahren für die Poisson-Gleichung $\triangle u = f$ auf dem Gebiet $\Omega = [0, 1]^2$ kennengelernt. Zur Erinnerung: Für jeden der $n \times n$ Gitterpunkte gilt eine Iterationsvorschrift, die den Einfluss der vier Nachbarpunkte auf die Unbekannte des aktuellen Punktes berücksichtigt, während für die Ränder Dirichlet-Bedingungen $u = const$ vorgegeben waren.

In dieser Aufgabe beschäftigen wir uns mit der Analyse der parallelen Eigenschaften des Verfahrens. Wir gehen dabei davon aus, das die Unbekannten in der üblichen kartesischen Reihenfolge nummeriert sind.

- Eine Parallelisierungs-Strategie, die wir schon implementiert haben, war das Aufteilen der Iterations-Matrix in Streifen ungefähr gleicher Länge. Führen Sie für diese Variante eine Isoeffizienz-Analyse durch und stellen Sie die Isoeffizienz-Funktion auf.
- Eine weitere Parallelisierungs-Strategie ist, die Iterations-Matrix in Blöcke aufzuteilen. Die Blöcke sollen nun in bewährter Schachbrett-Manier nummeriert sein. Führen Sie auch für diese Variante eine Isoeffizienz-Analyse durch mit aufstellen einer Isoeffizienz-Funktion.

Welche der beiden genannten Varianten ist nach Ihrer Analyse besser skalierbar?