INFORMATIK

Vorlesung + Übung Parallele Programmierung

Dozent: Peter Sanders

Umfang: 2 + 2 SWS

Vorlesung: Do 14-16, SR 013, Geb. 45. (21.5., 11.6: Feiertag)

Übung: ???

INFORMATIK

Literatur

- P. Sanders und T. Worsch, „Parallele Programmierung mit MPI – ein Praktikum“, ISBN 3-931216-76-4, Logos Verlag Berlin, 1997.
- V. Kumar, A. Grama, A. Gupta und G. Karypis, „Introduction to Parallel Computing. Design and Analysis of Algorithms“, Benjamin/Cummings, 1994.
- `http://www.mpi-sb.mpg.de/~sanders/parprog.html`
  - MPI Forum, „MPI: A Message-Passing Interface Standard“, `http://www.mpi-forum.org/docs/mpi-11.ps.Z`.
  - Folien
  - u.v.a.m.

INFORMATIK

Überblick

1. Parallele Maschinen und Modelle
2. Routingalgorithmen, Kostenmodelle, MPI Anfangsgründe
3. Zellularautomaten, Parallelisierunge duch Datenzerlegung
4. Latenzzeitverdeckung, Entkopplung, MPI Punkt-zu-Punktkommunikation
5. Parallele Matrizenrechnung, MPI kollektive Operationen
6. Lineare Gleichungssysteme, PDE, FFT
7. Sortieren

8./9. Lastverteilung I + II

10. MPI-2, Shared Memory
11. Komplexere Anwendungen

INFORMATIK

1 Aufbau von Parallelrechnern

1.1 Einzelne Prozessoren

1.1.1 Befehlspipeline

(siehe auch: J.L. Hennessy, D. A. Patterson, „Computer Architecture a Quantitative Approach“, 2. Auflage, 1996)

T. Ungerer, Mikroprozessortechnik, „TAT-Verlag“, 1995

1.1.2 Befehlsparallelismus

- Initiiere mehrere (z.B. 4) Befehle pro Takt (superskalar, VLIW, ...)
- Mehrere parallele Funktionseinheiten (z.T. mit Pipeline)
- Geht nur wenn Befehle sich nicht „in die Quere“ kommen.
- Sprungvorhersage wichtig
- Gute Compiler wichtig
- Trotzdem bleibt effiziente Programmierung schwer

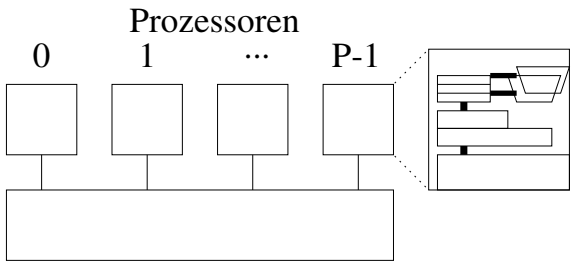
1.1.3 Speicherhierarchie

Ein (typischer?) 4-fach superskalarer 500MHz RISC-Prozessor.

	Verzögerung [s]	Bandbreite [Byte/s]	Kapazität [Byte]
Register	$2 \cdot 10^{-9}$	$64 \cdot 10^{9}$	512
L-1 Cache	$4 \cdot 10^{-9}$	$8 \cdot 10^{9}$	2 x 8–32K
L-2 Cache	$20 \cdot 10^{-9}$	$2 \cdot 10^{9}$	512K–4M
DRAM	$400 \cdot 10^{-9}$	$500 \cdot 10^{6}$	64M – 1G

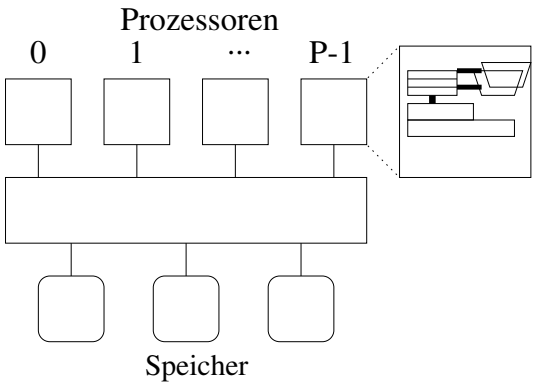
1.2 Typen von Parallelrechnern

1.2.1 Distributed Memory



- Sei P Anzahl Prozessoren (PE=Processing Element).
- PEs sind von 0 bis $P-1$ durchnumeriert

1.2.2 (Symmetric) Shared Memory



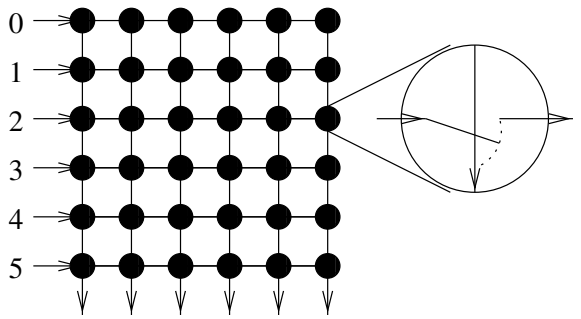
- Beispiel: Sun E10000
- Kann durch distributed memory emuliert werden indem Zugriffe auf nichtlokalen Speicher in Netzwerkkommunikationen umgesetzt werden. (Zum Beispiel SGI Origin2000)

1.2.3 Weitere Varianten

- Synchrone Befehlsabarbeitung (SIMD)?
- Vektorrechner: Eine (oder mehrere synchron arbeitende) Arithmetik-Pipeline + **Leistungsfähiges Speichersystem**
- Hybride Ansätze. Zum Beispiel Netzwerk von kleinen Shared-Memory-Maschinen (SMP), Shared Memory Maschine aus Vektorrechnern Mesh of Multistage-Networks, ...

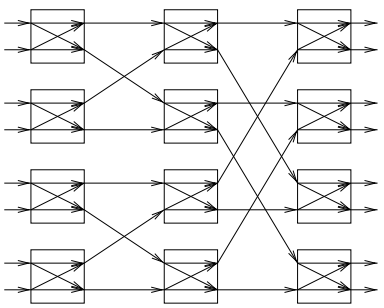
1.3 Verbindungsnetzwerke

1.3.1 Kreusschienenverteiler



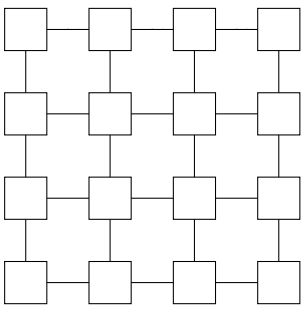
+ Schnell
+ Jede Permutation schaltbar
– P^2 Schalter nötig
- Beispiel (Sun E10000?)

1.3.2 Mehrstufig



+ Nur $\Theta(\log P)$ Schalter pro Pfad
+/– Insgesamt $\Theta(P \log P)$ Schalter
+/– Nachrichten können sich in die Quere kommen
– Lange Draehte, $\Theta\left(P^2\right)$ VLSI-Fläche, $\Theta\left(P^{3/2}\right)$ Volumen
- Beispiel: IBM-SP, SB-PRAM, (Cluster am MPII?)

1.3.3 Gitter



+ Kosten linear in P
– Lange Wege
– Nachrichten behindern sich massiv. (nur $P^{1/2}$ können gleichzeitig einen Schnitt überqueren)
- Beispiel: Intel Paragon, ASCI red (9000 PEs!); Cray T3D/T3E (3-dim)

Varianten:

- 1-dimensional, 3-dimensional
- Ring, Torus, Diagonalverbindungen, ...

1.4 Analyse Paralleler Algorithmen

- Vergleich mit seq. Ausführungszeit T_{seq}
- Bestimme parallele Ausführungszeit T_{par}
- Speedup $S := \frac{T_{\text{seq}}}{T_{\text{par}}}$
  - Absolut: T_{seq} ist „bestes“ sequentielles Programm
  - Relativ: T_{seq} ist paralleles Programm für $P = 1$
- Effizienz $E := \frac{S}{P}$
  - Häufiges Ziel: E =const
  - Sinnvolles Kostenmaß?
  - „Superlinearer“ Speedup ($E > 1$) möglich?

Einfaches Kostenmodell: Zeitbedarf für Nachricht der Länge m ist $T_{\text{start}} + mT_{\text{byte}}$

1.5 Warum Nachrichtengekoppelte Programmierung

Ziele:

- Effizienz
- Portabilität
- Leichte Programmierbarkeit

Alternativen:

- Direkter Hardwarezugriff
- Parallele Programmiersprache
- Unterprogrammbibliothek

1.6 Unterprogrammbibliotheken für Parallelverarbeitung

- Herstellereigen: ggf. schnell aber nicht portabel
- PARMACS hat sich nicht durchgesetzt
- PVM: einfach, portabel, kostenlos, flexibel. Geringer Funktionsumfang, manchmal langsam.
- MPI (Message Passing Interface)
  - \+ Normiert
  - \+ Portabel
  - \+ Relativ Effizient
  - \+ Guter Kompromiss zwischen Einfachheit und hohem Funktionsumfang

1.7 Überblick über MPI

- Punkt-zu-Punkt-Kommunikation
- Kollektive Operationen. Beispiel: Broadcast
- (Datentypen)
- (Gruppen, Kontexte, Kommunikatoren)
- (Virtuelle Topologien)
- (MPI-2: Gemeinsamer Speicher, Prozesse, I/O, ...)
- (Was fehlt: Multithreading, ...)