Fahrplan

1. Parallelrechner vs. (heterogener) Cluster
2. Kommunikationsschemata
3. Paralleles Programmieren
3.1. Paradigmen
3.2. Kommunikation (Message Passing)
3.3. Effizienz
4. Message Passing Systeme

"Es (das parallele Programmieren) ist die bereits greifbare Chance, die fast 50 Jahre währende Knechtschaft des sequentiellen Denkens und Programmierens, Testens und Debuggens endlich abstreifen und in eine n-dimensionale Freiheit der parallelen Ausdrucksmöglichkeiten entfliehen zu können."

1. Parallelrechner vs. (heterogener) Cluster

"echter Parallelrechner"	**Cluster**
+ hohe Rechenleistung der Knoten + hohe Kommunikationsleistung - vorgegebene Topologie - schlecht skalierbar + koherentes System - teuer	+ hohe Rechenleistung der Knoten - geringe Kommunikationsleistung + frei wählbare Topologie + sehr gut skalierbar - u.U. heterogenes System + preiswert + Parallelrechner können Knoten sein

2. Kommunikationsschemata

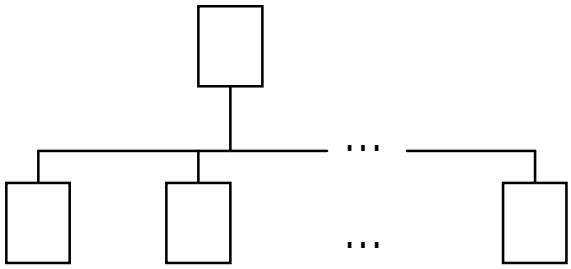
shared memory	**message passing**
+ schnell - komplizierter Aufbau - distributed shared memory → Datenaustausch über Netzwerk	- langsam - Fehlertoleranz + einfacher Aufbau
Convex SPP, Cray T3D, Meiko CS2, Nec Cenju-3, SGI Power Challenge	Transputer, WS-Cluster, Intel iPSC/2, IBM SP/2, nCube, TMC CM-5

- message passing kann im shared memory emuliert werden
- andere Richtung prinzipiell auch möglich, aber eindeutig zu langsam

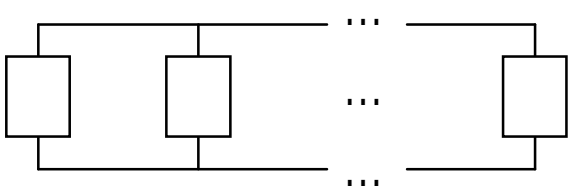
3. Paralleles Programmieren

3.1. Paradigmen

master-slave



node-only



crowd computation

- Initialisierung
- Berechnung
- Einsammeln der Ergebnisse

Tree

- Prozeß erzeugt Teilaufgaben und verteilt diese auf neue Prozesse (spawning)

Workload Allocation

- Data Decomposition, Daten werden verteilt (→ XIMD)
- Function Decomposition, verschiedene Teilaufgaben (→ MIXD)

3.2. Kommunikation (Message Passing)

nicht blockierend/blockierend

- non-blocking
  Sende-/Empfangsvorgang wird angestoßen
  eine Komplettierungsanweisung realisiert eine Blockierung

- blocking
  Sendung beendet, sobald Daten vollständig abgesetzt
  Emfang einer Nachricht vollständig beendet
  → dead-lock möglich

- synchron
  Sendung erst beendet, wenn Empfang bestätigt (Rendevouz)

- ready
  Sendung beginnt erst, wenn Empfang gestartet wurde

Point-to-Point Kommunikation

- ein Sender schickt Daten an einen Empfänger

collective communication

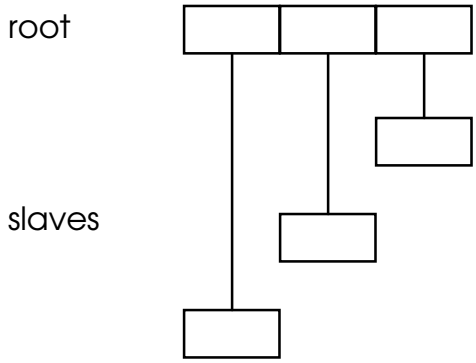
- ein oder mehrere Prozesse kommunizieren gleichzeitig miteinander

- barrier synchronization
  alle Prozesse einer Gruppe warten an einer Stelle, bis alle anderen ebenfalls dort eingetroffen sind

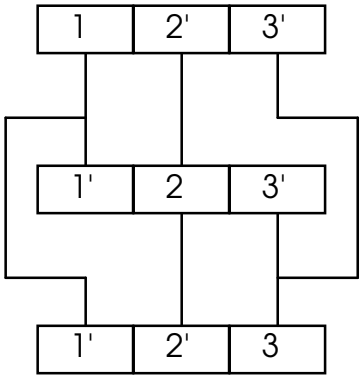
- broadcasting
  ein Prozeß sendet eine Nachricht an alle Prozesse einer Gruppe

- gathering

zwei unterschiedliche Interpretationen:
ein Prozeß sammelt alle Daten einer Gruppe



alle Prozesse sammeln alle Daten (gather-from-all)



- scatter

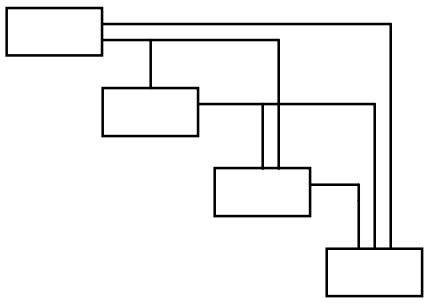
Daten werden von einem Prozeß an alle anderen verteilt
ein scatter-to-all ist gleichzusetzen mit gather-from-all

- global reduction

eine Operation wird auf eine Menge verteilter Daten angewandt
(z.B. Summenbildung, Maximum, Minimum)

- scan

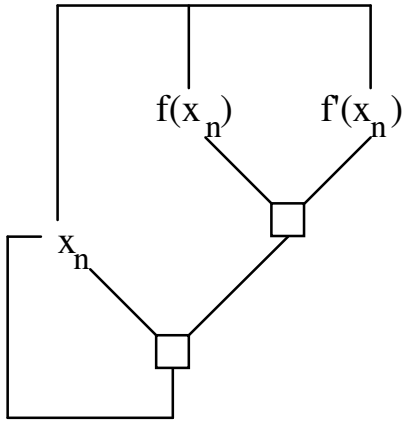
führt eine hierarchische Reduktion innerhalb einer Gruppe



Synchronisation von Prozesseinheiten

- klassisches Problem:

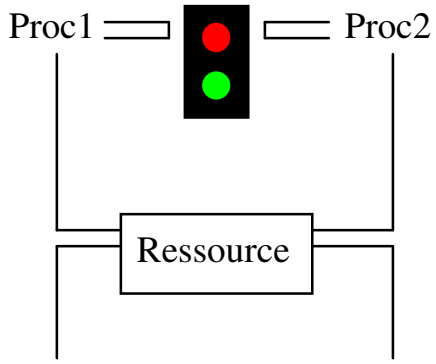
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



P_1: r_1:=f(x) || P_2: r_2:=f'(x)
sync(P_1,P_2)
x:=x-r_1/r_2

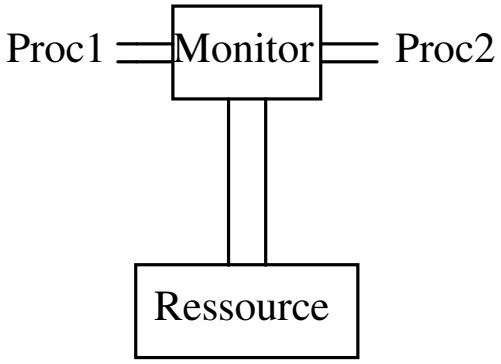
- Ausschlußsynchronisation (bei Zugriff auf gemeinsame Ressourcen)

semaphor



günstig im shared memory zu realisieren

monitor



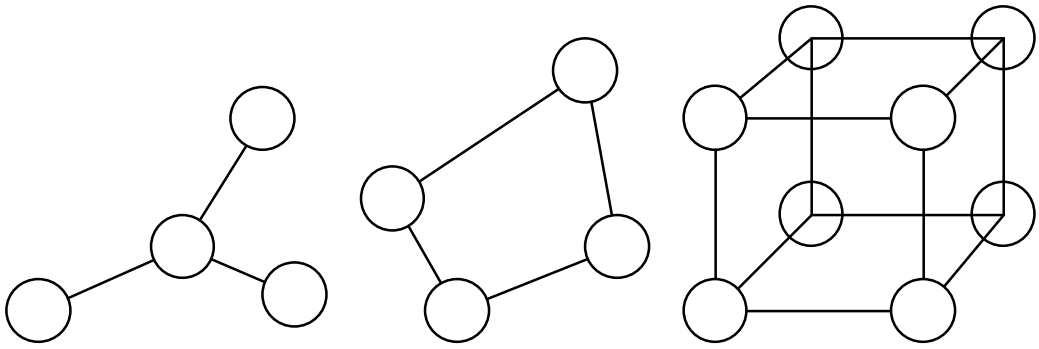
Monitor wird als eigenständiger Prozeß ausgeführt

Processor Sets

bezeichnen eine Menge von Prozeßeinheiten, die einander kennen.
Zusammen mit einer Verbindungsstruktur entsteht eine Topologie.

Topologien

beschreiben, wie Prozeßeinheiten miteinander verbunden sind, d.h. welcher Prozessor mit welchem direkt kommunizieren kann.
Man unterscheidet Stern- und Ringtopologien, Toren, Hypercubes u.a.



Die Anzahl der Kanten, die auf dem kürzesten Weg zwischen Knoten A und Knoten B liegt wird "hops" genannt. Je weniger hops, desto schneller kann ein Nachrichtenaustausch stattfinden.

Es ist günstig, die Topologie and das Problem bzw. umgekehrt anzupassen. Beispiel:

> Über einem Gitter soll eine Iteration mit einer teuren Funktionsauswertung stattfinden, die die Werte ihrer Nachbarn benutzt, also etwa

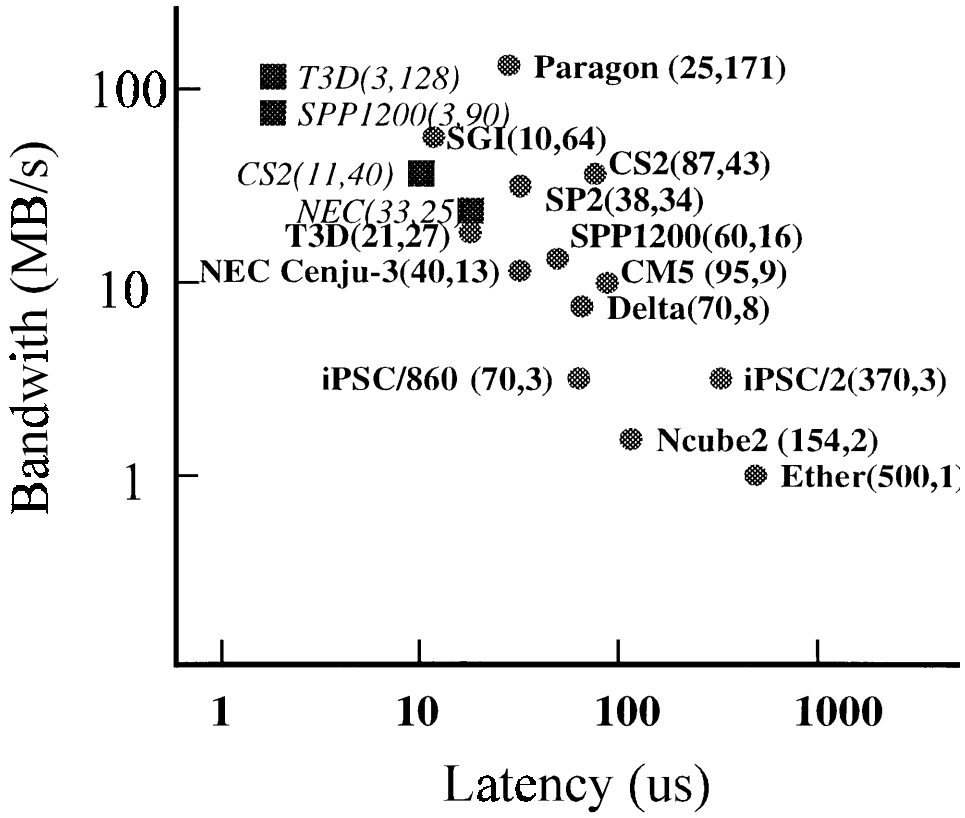
$$f_{i,j}^{n+1} = \Phi\left(f_{i,j}^{n}, f_{i-1,j}^{n}, f_{i+1,j}^{n}, f_{i,j-1}^{n}, f_{i,j+1}^{n}\right)$$

> hierfür scheint die Parallelisierung auf einem Stern sinnvoll.

3.3. Effizienz

Zeit für eine Nachrichtenübermittlung $t_n = \alpha + n\beta + (h-1)\gamma$ (DONGARRA)

- α - startup-time
- β - cost per byte
- γ - hop-delay
- n - bytes per message
- h - number of hops



4. Message Passing Systeme

UNIX V IPC

semaphore
messages
shared memory
sockets (internet, UNIX-domain)

PVM

(IPC Sockets) (Geist, Beguelin, Dongarra, Jiang, Manchek, Sunderam)

MPI

(IPC Sockets) (Geist, Dongarra, Gropp, Lusk, Skjellum)
Effizienzvergleich von Andreas

MPI, für MPP Systeme	PVM, virtual machine
+ MPP + point-to-point Kommunikation + collective Operationen +Topologie + save communicatio	+ heterogener Cluster + Fehlertoleranz + dynamische Konfigurierbarkeit + Prozeßkontrolle - overhead durch Architekturbestimmung

p4

Linda

PARMACS

Chameleon

- Aufsatz und einheitliches Interface auf Kosten Funktionsumfang
- realisiert komplexe Operationen

helloworld
mamult (dynamic workload)

intervallschachtelung oder quick-sort (Baum-Topologie)

n-particle(kreis-topologie)
    (gather)
    (scatter)