3 Einfache Parallele Algorithmen

Überblick

- Work-Time Darstellung von Algorithmen
- Gitterförmige Verbindungsnetzwerke
- Parallelisierung durch Datenzerlegung (Beispiel: Zellularautomaten)

3.1 Work-Time Darstellung

[JáJá 1.5]

1. Beschreibe Algorithmus als Abfolge von Zeitschritten in denen unabhängige Operationen durchgeführt werden können

   W_i: Anzahl Operationen in Schritt i (Anzahl benötigter Prozessoren um es in einem Schritt zu machen)

   T: Anzahl Zeitschritte

2. Scheduling der einzelnen Operationen auf p Prozessoren, so daß $\lceil W_i/p \rceil$ PRAM Schritte ausreichen. $\Rightarrow$ Gesamtzeit $T_p \leq \sum_i W_i + T$

Beispiel: Summe Berechnen

n Eingaben

- $W_i = n/2^i$
- $T = \log n$
- physikalischer Prozessor i simuliert logische Prozessoren $in/p \ldots (i+1)n/p - 1$.
- $T_p = \sum_{i=0}^{\log n} \left\lceil \frac{n}{p2^i} \right\rceil \approx \frac{n}{p} + \log p$

Diskussion

+ Abstrakter, p im ersten Schritt uninteressant
+ Man gewinnt gleich eine ganze Klasse von Verfahren (von schnell bis effizient)
+ Scheduling oft einfach und wiederverwendbar
– Scheduling i.allg. nicht trivial $\Rightarrow$ Load Balancing

- Vergleich von Algorithmen:
  - Schnell versus langsam (nur $T(n)$)
  - Wie effizient? (maximales effizientes p) [Kumar]
- Bei Verbindungsnetzwerken zusätzlich Kommunikationsaufwand bzw. Lokalität beachten. $\Rightarrow$ Scheduling nicht immer so einfach, zusaetzliche Terme.

3.2 Gitterförmige Verbindungsnetzwerke

[Leighton 1][Kumar 2.4]

Allgemein d Dimensionen $G = (V, E)$

$$V = \{0..n_1 - 1\} \times \cdots \times \{0..n_d - 1\}$$

$$(\mathbf{i}, \mathbf{j}) \in E \subseteq V \times V$$

gdw. $\|\mathbf{i} - \mathbf{j}\|_1 = 1$

Theorie: Meist $n_1 = \cdots = n_d = p^{1/d}$.

Verallgemeinerung

- Ring,Torus, ...
- Diagonalverbindungen

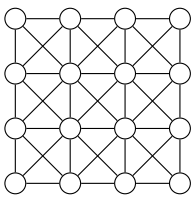
Diskussion $d \leq 3$

+ Drahtlänge unabhängig von p
+ konstanter Knotengrad
+ Bei 2D keine Kühlprobleme
+ Einfach
– Relativ hoher Durchmesser
– Geringer Durchsatz bei globaler Kommunikation $\rightarrow$ Bisektionsbreite

4 Beispiel: Simulation von Zellularautomaten

Zellularautomat $(Q, 0, N, \delta, d)$

- Endliche Zustandsmenge Q
- Ruhezustand 0
- Nachbarschaft (Raster) N. Zum Beispiel M_1: Eine Zelle und ihre 8 Nachbarn in horizontaler und diagonaler Richtung. $(c, n, o, s, w, no, so, sw, nw)$
- Überführungsfunktion $\delta : Q^{|N|} \rightarrow Q$.
- Dimension: 1, 2, 3, (4), ...



Arbeitsweise

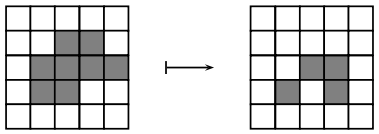
- synchron agierende
- identische
- kommunizierende
- endliche Automaten

In einem Zeitschritt führen alle endlichen Automaten synchron einen Zustandsübergang aus.

4.0.1 Eine (Fast-)Majoritätsregel

- $Q = \{0, 1\}$
- $d = 2$
- $N = M_1$ Moore-Nachbarschaft
- $\delta(c, n, o, s, w, no, so, sw, nw) =$
  $f(c + n + o + s + w + no + so + sw + nw)$

n	0	1	2	3	4	5	6	7	8	9
$f(n)$	0	0	0	0	1	0	1	1	1	1



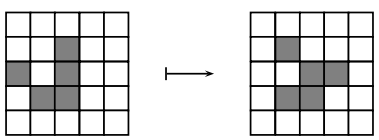
Ein Modell für "Entmischung"

Huebsche Bilder die der Drucker nicht mag...

LIFE

$$\delta(q_{\mathbf{c}}, q_{\mathbf{n}}, q_{\mathbf{o}}, q_{\mathbf{s}}, q_{\mathbf{w}}, q_{\mathbf{no}}, q_{\mathbf{so}}, q_{\mathbf{sw}}, q_{\mathbf{nw}})$$
$$= g(q_{\mathbf{c}}, q_{\mathbf{n}} + q_{\mathbf{o}} + q_{\mathbf{s}} + q_{\mathbf{w}} + q_{\mathbf{no}} + q_{\mathbf{so}} + q_{\mathbf{sw}} + q_{\mathbf{nw}}) \ .$$

n	0	1	2	3	4	5	6	7	8
$g(0,n)$	0	0	0	1	0	0	0	0	0
$g(1,n)$	0	0	1	1	0	0	0	0	0



Überführungsfunktion

```
/* transition table for the annealing rule */
static State anneal[10] =
   {0, 0, 0, 0, 1, 0, 1, 1, 1, 1};

/*  compute the new state of a cell */
#define transition(a, x, y)
   (anneal[(a)[(y)-1][(x)-1] + \
           (a)[(y)  ][(x)-1] + \
           (a)[(y)+1][(x)-1] + \
           (a)[(y)-1][(x)  ] + \
           (a)[(y)  ][(x)  ] + \
           (a)[(y)+1][(x)  ] + \
           (a)[(y)-1][(x)+1] + \
           (a)[(y)  ][(x)+1] + \
           (a)[(y)+1][(x)+1]   \
          ])
```

Idee: C++ mit `inline` Funktion.

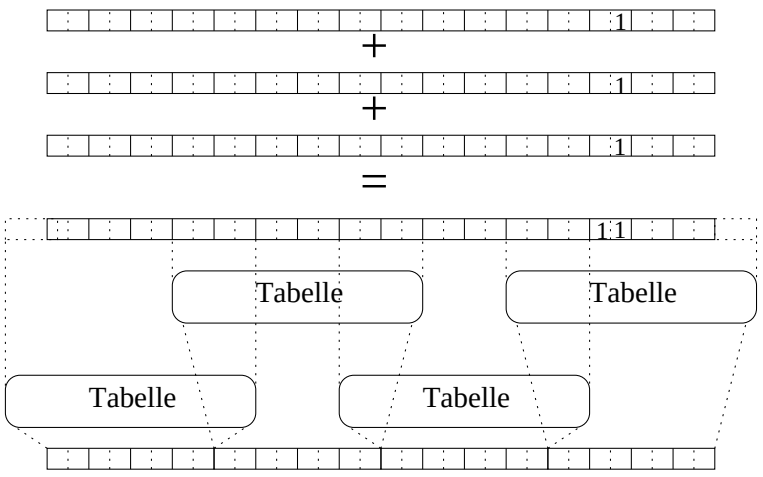
Sequentielle Simulation

```
/*  simulate one step of the CA
 *  IN:
 *  from:       pointer to old configuration
 *  to:         pointer to new configuration
 *  configRows: number of rows of configurations
 *  OUT:
 *  to:         the memory is modified
 */
static void
simulate(Row *from, Row *to, int configRows)
{
   int x,y;

   boundary(from, configRows);
   for (y = 1;  y <= configRows;  y++) {
      for (x = 1;  x <= CONFIGCOLS;  x++) {
         to[y][x  ] = transition(from, x  , y);
      }
   }
}
```


Periodische Randbedingungen

```
/*  treat boundary conditions
 *  IN:
 *  buf:       pointer to a stored configuration
 *  configRows:number of rows of configurations
 *  OUT:
 *  buf:       configuration is modified in plac
 */
static void boundary(Row *buf, int configRows)
{ int x,y;

  for (y = 1;  y <= configRows;  y++) {
    buf[y][0           ] = buf[y][CONFIGCOLS];
    buf[y][CONFIGCOLS+1] = buf[y][1         ];
  }

  for (x = 1;  x <= CONFIGCOLS;  x++) {
    buf[0           ][x] = buf[configRows][x];
    buf[configRows+1][x] = buf[1         ][x];
  }
}
```


4.0.2 Bitparallele Simulation



Vergleich

- Parallele Addition für 16 Zellen
- Paralleler Lookup für 4 Zellen
- 4-fach geringerer Speicherbedarf (Cache?)
- 4-fach mehr Zellen pro Cache-miss
- 16-fach mehr Zellen pro Load- (Store-) Befehl
- Pentium-II-300: 20M Zellen/s $\Longrightarrow$ 160M Zellen/s (512×512 Zellen)

Umsetzung

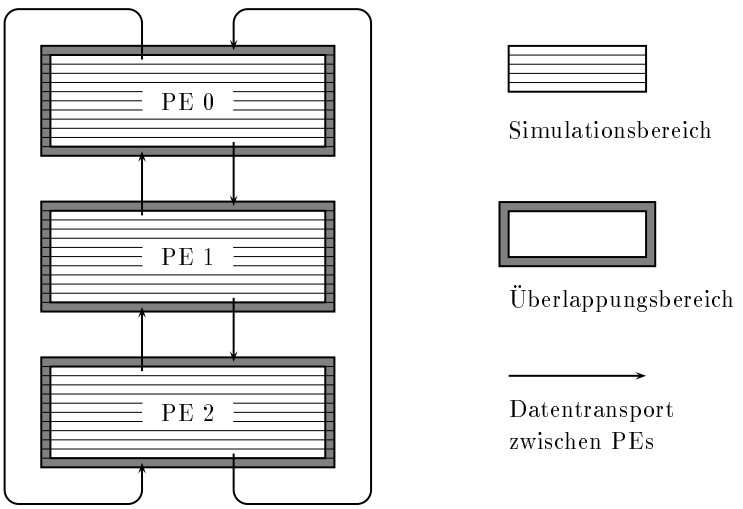
```
static void simulate(Line *from, Line *to, int lines)
{ int x, y;
  unsigned int l, m, r;
  boundary(from, lines);
  for (y = 1;  y <= lines;  y++) {
    l = from[y-1][0] + from[y][0] + from[y+1][0];
    m = from[y-1][1] + from[y][1] + from[y+1][1];

    for (x = 1;  x <= XSIZE; x++) {
      r=from[y-1][x+1]+from[y][x+1]+from[y+1][x+1];
      to[y][x] =
        (transition[((l& 3u        )<<10)|
                    ((m&(1023u<<22))>>22) ]<<24)|
        (transition[((m&(4095u<<14))>>14) ]<<16)|
        (transition[((m&(4095u<< 6))>> 6) ]<< 8)|
         transition[((m& 1023u     )<< 2)|
                    ((r&(   3u<<30))>>30)  ];
      l = m; m = r;
    }
  }
}
```


Weitere Maßnahmen:

- 64 Bit
- Paralleler Lookup für 8 Zellen. Wirklich?
- 1 Bit pro Zustand
  - − δ als boolesche Funktion `& | ^ ~`. (LIFE)
  - + 2–16 mal mehr Parallelismus
  - + Keine Tabellenzugriffe
  - − Viele Instruktionen
- Ausrichtung nach Cache-Line-Grenzen
- Prefetching, Zeilenpuffer
- MMX, Grafikhardware?
- Einfachere Behandlung stationärer Bereiche. Datenstrukturen ...

4.1 Parallelisierung durch Datenzerlegung

4.1.1 Streifenweise Zerlegung



Ein einfacher Ansatz:

- Streifenweise (anderes folgt)
- Alle PEs gleich viel. Zwei Bedingungen?
- Nur Randbehandlung wird geändert

Neue Randbehandlung

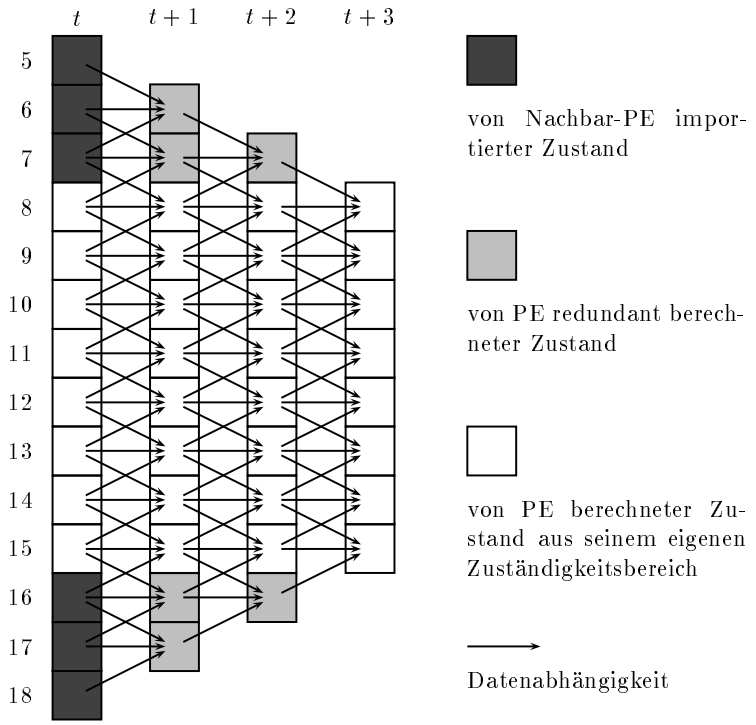
```
#define STATE MPI_CHAR

static void yBoundary(Line *buf, int lines)
{ MPI_Status dummy;

  /* send down */
  MPI_Sendrecv(buf[lines], XSIZE + 2, STATE,
               (iProc + 1        ) % nProc, 42,
               buf[0    ], XSIZE + 2, STATE,
               (iProc - 1 + nProc) % nProc, 42,
               MPI_COMM_WORLD, &dummy);

  /* send up */
  MPI_Sendrecv(buf[1]        , XSIZE + 2, STATE,
               (iProc - 1 + nProc) % nProc, 43,
               buf[lines + 1], XSIZE + 2, STATE,
               (iProc + 1        ) % nProc, 43,
               MPI_COMM_WORLD, &dummy);
}
```


Analyse

Zeit pro Iteration:

- $T_{\text{seq}} \approx n^2 T_{\text{sim}}$
- $T_{\text{par}} \approx \frac{n^2}{P} T_{\text{sim}} + 4(T_{\text{start}} + \frac{n}{4} T_{\text{byte}})$
- $T_{\text{sim}} \approx 6 \cdot 10^{-9}\text{s}$
- $T_{\text{start}} \approx 300000 \cdot 10^{-9}\text{s}$
- $T_{\text{byte}} \approx 30 \cdot 10^{-9}\text{s}$
- $E = \frac{T_{\text{seq}}}{P T_{\text{par}}} > 1/2$???

4.1.2 Entkopplung von Berechnungen



4.1.3 Analyse

- Überlappungsgröße k
- k Iterationen:
  $T_k = kT_{\text{sim}}(m + k - 1)n + 4(T_{\text{start}} + \frac{n}{4} kT_{\text{byte}})$
- $kT_{\text{par}} - T_k = kT_1 - T_k =$
- $(k-1)(4T_{\text{start}} - knT_{\text{sim}})$
- Ab $2T_{\text{start}} > nT_{\text{sim}}$ lohnt sich $k \geq 2$
- Welches k ist optimal? Minimiere T_k/k
- D.h. $\frac{\partial}{\partial k} T_{\text{neu}} = 0$
- $k_{\text{opt}} = 2\sqrt{\frac{T_{\text{start}}}{nT_{\text{sim}}}}$
- Wann wird jetzt gute Effizienz erreicht?

Diskussion

- $T_{\text{par}} \approx \frac{n^2}{P} T_{\text{sim}} + \mathbf{8}(T_{\text{start}} + \frac{n}{4\sqrt{\mathbf{P}}} T_{\text{byte}})$
- Gut wenn $nT_{\text{byte}} \gg T_{\text{start}}$
- \+ Geringer Kommunikationsumfang vor allem bei großem P.
- \+ Weniger redundante Arbeit bei Entkopplung
- – „Magische Prozessorzahlen“
- – Spaltenaustausch ist kompliziert (MPI Datentypen helfen etwas, Entkopplung hilft)
- Nichquadratische Teile gut?

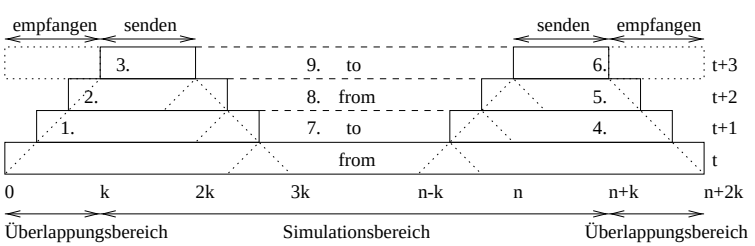
4.1.4 Kommunikationszeitverdeckung zur Zellularautomatensimulation

Hauptschleife:

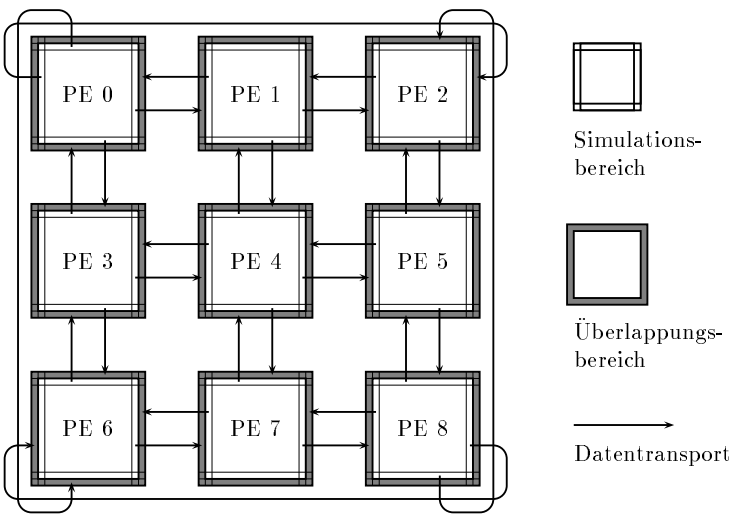
- Simuliere Zeilen 1 und n
- Starte Kommunikation für die nächste Iteration
- Simuliere Zeilen 2 bis $n-1$
- Warte auf Komplettierung der der Kommunikation

4.1.5 Entkopplung + Überlappung

Procedure simulate(x, y, k) (* k odd *)
for $i := k-1$ **downto** 0 **do**
  simulate lines $k-i$ through $2k+i$, $x \to y$
  simulate lines $n-i$ through $n+k+i$, $x \to y$
  swap x and y
start send lines $x[k]$ through $x[2k-1]$ up
start send lines $x[n]$ through $x[n+k-1]$ down
start receive lines $y[0]$ through $y[k-1]$ from above
start receive lines $y[n+k]$ through $y[n+2k-1]$ from be
for $i := 0$ **to** $k-1$ **do**
  simulate lines $3k-i-1$ through $n-k+i$, $y \to x$
  swap x and y
complete communication



Zweidimensionale Problemzerlegung



Blick über den Tellerrand

- Anwendungen
  - Stömungssimulation, Spin-Gläser, Bildverarbeitung
  - Explizite iterative Lösung diskretisierter partieller Differentialgleichungen
  - Zeitabhängige partielle Differentialgleichungen
  - Partikelsimulation
- Weitere Algorithmische Probleme
  - Lastverteilung
  - Komplizierte Randbedingungen
  - Unregelmäßige Gitter, Schrittweiten und Überführungsfunktionen
  - Variable Gitter
  - Pyramiden bzw. Multigrid

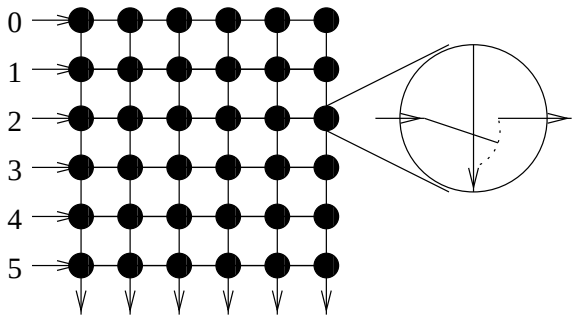
5 Verbindungsnetzwerke

Überblick

- Jeder mit Jedem
- Hyperwürfel
- Butterfly
- Hierarchische Verbindungsnetzwerke
- Einbettung
- Maßzahlen und Vergleich

5.1 Kreuzschienenverteiler

(Crossbar)



+ Schnell
+ Jede Permutation schaltbar
– P^2 Schalter nötig, (hier = VLSI-Aufwand)
- Beispiel (Sun E10000?)
- Graphentheoretisch: Prozessoren bilden eine Clique

5.2 Hyperwürfel

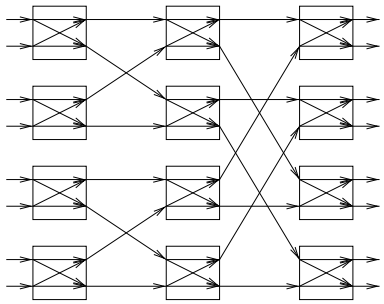
Sei $p = \{0,1\}^d \times 0..d-1$:

Hyperwürfel = d-dimensionales Gitter der Seitenlänge zwei.

(Was wenn man einen Torus daraus macht?)

+ Schön symmetrisch (Was heisst das?)
+ Koordinaten kann man einfach als Binärzahl lesen
+ kleiner Durchmesser
+ Viele interessante Algorithmen haben die passende Struktur (Beispiel: Summenberechnung)
+ Hohe Bandbreite
+ Das „universelle“ Gitternetzwerk
– Kein konstanter Knotengrad
– Lange Drähte, hoher VLSI Aufwand
– p ist nicht beliebig

5.3 Butterfly



$V = 0..2^d - 1 \times 0..d$ (row $\times$ level)

$E = \{< (i,j),(i,j+1) >: 0 \le i < 2^d, 0 \le j < d\} \cup$
$\{< (i,j),(i \oplus 2^{d-j}, j+1) >: 0 \le i < 2^d, 0 \le j < d\}$
und Rückwärtskanten ($\oplus$ = exklusiv oder)

+ Nur $\Theta(\log P)$ Schalter pro Pfad

+/− Insgesamt $\Theta(P \log P)$ Schalter

+/− Nachrichten können sich in die Quere kommen

- Beispiel: IBM-SP, SB-PRAM, (Cluster am MPII?)

Varianten und Sichtweisen

- Links Prozessoren rechts Speichermodule
- Prozessoren mit linker und rechter Spalte verbunden
- Jeder Knoten ein Prozessor
- Eine Butterfly-Zeile $\leftrightarrow$ Hyperwürfelknoten (Was heisst das?)
- Etwas abstrakter, *Fat Tree*: Prozessoren nur bei Schicht d; identifiziere Knoten (i,j) mit j gemeinsamen least significant bits

5.4 Sonstiges

Weniger geeignete Netzwerke

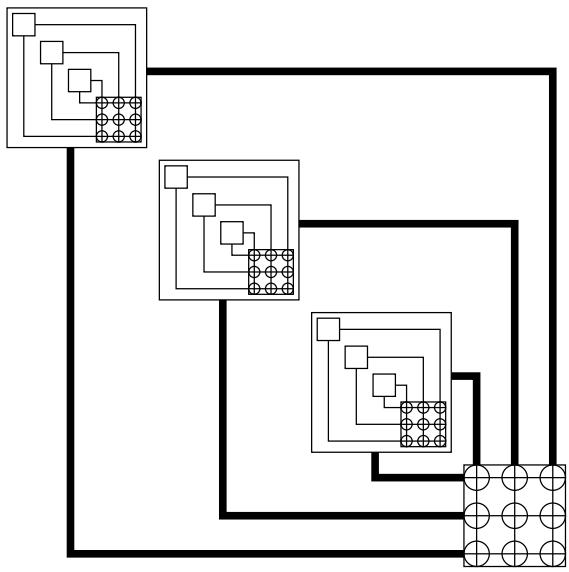
- Stern
- Baum (Warum?)
- Bus

Hybride Netzwerke

Fass kleine Netzwerke zu Metaknoten zusammen und bilde daraus Netzwerke

+ Die kleinen Netzwerke müssen nicht skalierbar sein
+ Für die Metaknotenverbindungen kann man sich aufwendigere Technologien leisten (Warum?). Beispiel: Glasfaser
+ Mix-and-match von off-the-shelf Technologien. Beispiel: Cluster am MPII.

5.5 Hierarchischer Kreusschienenverteiler



- Beispiel: NEC SX-5, Cray SV-1, Hitachi?, (Sun E10000), MPII-Cluster, DEC/Compaq

5.6 Kombination mit Speicheerhierachien

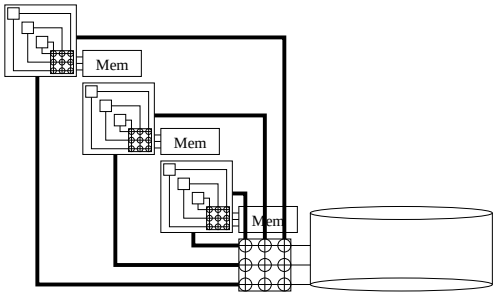


Abbildung von Speicherhierarchieebenen auf die Ebenene der Netzwerkhierachie Map memory hierachy levels to network hierarchy levels.

Beispiel:

- Register und L-1 cache lokal
- On-chip-Speicher geteilt von 4 Prozessoren auf einem Chip
- Hauptspeicher von 4×4 Prozessoren auf einem Bord geteilt
- 64 Festplatten geteilt von $8 \times 4 \times 4$ PEs in einem Schrank
- 16 Schränke
  verbunden über ein optisches Netzwerk teilen sich einen Bandroboter

5.7 Einbettung

Gegeben zwei Graphen $G = (V, E)$ und $G' = (V', E')$. Eine Einbettung wird definiert durch eine Knotenabbildung

$$f : V \to V'$$

und eine Kantenabbildung

$$g : E \to E'^*$$

von Kanten von G auf Pfade in G'.

Knotenlast: $\max_{u \in V'} \left|f^{-1}(u)\right|$

Kantenlast:
$\max_{e \in E'} |\{P = (e_1, \ldots, e, \ldots, e_k) : P \in \text{Bild}(g)\}|$

Dilation: $\max_{e \in E} |g(e)|$

Ein-zu-Eins-Einbettung: f ist bijektiv. Oft identifiziert man dann V und V'

5.7.1 Beispiel: Summenbildung auf 1D-Gitter

Knotenlast: $\log p$

Kantenlast: $\log p$

Dilation: $p/2$

Aber, der Teilgraph jedes Berechnungsschritts hat Knoten- und Kantenlast 1!

5.8 Dimension-Order-Einbettung

Eins-zu-Eins-Einbettung der Clique
$K = (V = \{0..n-1\}^d, V^2)$ in ein n^d-Gitter G.
$g((v_1, \ldots, v_d), (u_1, \ldots, u_d)) =$
$[(v_1, \ldots, v_d), \ldots,$
$(u_1, \ldots, v_d), \ldots,$
$(u_1, u_2, \ldots, v_d), \ldots,$
$(u_1, \ldots, u_d)]$ „Passe eine Koordinate nach der anderen an“

Satz: Kantenlast ist $\leq n^{d+1}/4$.

Beweis für $d = 2$. Geht es besser?

5.9 Maßzahlen

5.9.1 Durchmesser

Längster kürzester Pfad

$$\text{diam}(G) := \max_{v \in V, w \in V} \min_{(v,\ldots,w)} |(v, \ldots, w)|$$

Bisektionsbreite

Wieviele Kanten werden mindestens zerschnitten wenn man das Netzwerk in zwei Hälftern teilt

$$\min_{S \subseteq V, |S| = \lceil |V|/2 \rceil} |E \cap (S \times (V - S))|$$

5.9.2 Beispiel

Eine Clique hat die Bisektionsbreite $p^2/2$ (p gerade).

Untere Schranken für die Bisektionsbreite

Satz: Sei (i, g) eine Eins-zu-Eins-Einbettung einer Clique in einen Graphen $G = (V, E)$, mit Kantenlast L dann ist die Bisektionsbreite von G mindestens

$$\frac{|V|^2}{2L}$$

Korollar: Ein n^d-Gitter (n gerade) hat Bisektionsbreite $2n^{d-1}$.

Korollar: Ein Hyperwürfel hat Bisektionsbreite p.

5.10 Zufallfgraphen als Verbindungsnetzerk

Sei $G = (V, E) \in \mathcal{G}(p, c/p)$, c eine geeigneter Konstante, d.h., jede *mögliche* Kante (u, v) wird mit Wahrscheinlichkeit $q = c/p$ ausgewählt.

Satz: Sei p gerade. Für $c \geq 7$ gehen aus jeder Knotenmenge der Größe $W \geq p/2$ mindestens $p/2$ Kanten heraus mit Wahrscheinlichkteit mindestens $1 - e^{-\mathcal{O}(p)}$.

Beweis: Sei $w(S)$ die Anzahl Kanten, die eine gegebene Menge von Knoten S verlassen.

$$\begin{aligned}
\mathbf{Pr}[W > p/2] &= \mathbf{Pr}[\exists S \subseteq V, |S| = p/2 \,|\, w(S) \leq p/2] \\
&\leq \binom{p}{p/2} \mathbf{Pr}[w(S) \leq p/2] \text{ festes } S, |S| = p/2 \\
&\leq 2^p \mathbf{Pr}[w(S) \leq p/2] \\
&\leq 2^p e^{-(1-2/c)^2 cp/4} \text{ (Chernoff Schranke)} \\
&= e^{(\ln 2 - (1-2/c)^2 c/4)p}
\end{aligned}$$

Das is exponentiell klein falls

$$(1-\frac{2}{c})^2\frac{c}{4} > \ln 2$$

Eine Chernoff-Schranke

Satz: Seien $X_1, \ldots, X_n$ unabhängige identisch verteilte 0-1 Zufallsvariablen mit $\mathbf{Pr}[X_i = 1] = q$. Sei $Y := \sum_{i=1}^{n} X_i$. Dann gilt für $0 < delta < 1$

$$\mathbf{Pr}[Y < (1-\delta)qn] \leq e^{-\delta^2 qn/2}.$$

6 Kollektive Operationen

[Kumar Kapitel 3]

Überblick

- Broadcast
- Reduktion
- Gather/Scatter
- Prefix-Summe
- Gossiping
- All-to-all Personalized Communication
- Unregelmäßige Nachrichtenlängen

Nachrichtenlänge n Bytes.

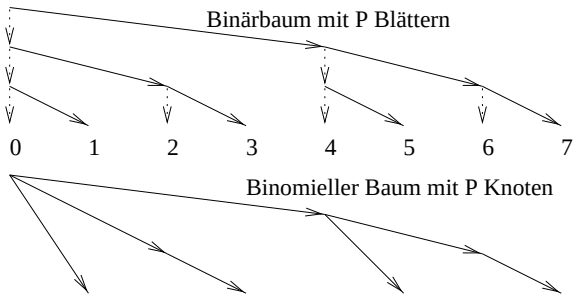
6.1 Broadcast

Ein PE (z.B. No. 0) schickt Nachricht an alle anderen.

6.1.1 Kurze Nachrichten

Algorithmus wie Summenbildung

- $T_{\text{broadcast}}(p,n) \approx (T_{\text{start}} + nT_{\text{byte}}) \log p$
- Optimal für kleine n und synchrone single-ported Kommunikation. (Schiefe Bäume besser falls letztere Einschränkung nicht gilt.)
- Klappt auch auf Gittern mit Hardware-Router
- Verallgemeinerung: Mehrere unabhängige Broadcasts auf unabhängigen Teilen des Verbindungsnetzerks.



6.1.2 Einfache Pipeline

Nachricht wird in k Stücke zerlegt

PE i
  for $j := 1$ **to** k **do**
    if $i \geq 1$ **then** receive piece j from PE $i-1$
    if $i \leq p-2$ **then** send piece j to PE $i+1$

$$T_{\text{broadcast}}(p,n) = (T_{\text{start}} + \frac{n}{k} T_{\text{byte}})(2k + p - 3)$$

optimales k:

$$k = \sqrt{\frac{n(p-3)T_{\text{byte}}}{2T_{\text{start}}}}$$

dann ist

$$T_{\text{broadcast}}(p,n) \approx 2nT_{\text{byte}} + pT_{\text{start}} + \sqrt{2npT_{\text{start}}T_{\text{byte}}}$$

Gut wenn $nT_{\text{byte}} \gg pT_{\text{start}}$. Sonst schmerzt die große Zahl Startups.

6.1.3 Broadcast auf Bussen

Wirklich trivial?

6.1.4 Pipelined Tree Broadcast

Gegeben ein binärer Baum
(z.B. vollst. Baum aber es geht besser)

for $j := 1$ **to** k **do**
  if parent exists **then** receive piece j from parent PE
  if left child exists **then** send piece j to left child PE
  if right child exists **then** send piece j to right child P

$$T_{\text{broadcast}}(p,n) \approx (T_{\text{start}} + \frac{n}{k} T_{\text{byte}})(3k + d)$$

wobei d die Anzahl Schritte ist, nach denen das erste Paket alle Knoten erreicht hat.

Optimales k:

$$k \approx \sqrt{\frac{ndT_{\text{byte}}}{T_{\text{start}}}}$$

dann ist

$$T_{\text{broadcast}}(p,n) \approx 3nT_{\text{byte}} + dT_{\text{start}} + \mathcal{O}(\sqrt{ndT_{\text{start}}T_{\text{byte}}})$$

- Für große n max 50% langsamer als einfache Pipeline
- $d \approx \log_\phi p$ bei geschickter Wahl des Baums. ($\phi = (1+\sqrt{5}/2) \approx 1.6$)
- Für kleine n max 30% langsamer als der einfache Alg. für vollständige Binärbäume.

– Funktioniert nicht für 1D-Gitter

\+ Aber für 2D (3D) Gitter mit Hardware-Router (Wie?)

6.1.5 Große aber nicht sehr große n

Ein baumartiges Kommunikationsmuster löst das Problem in Zeit $(2+\epsilon)nT_{\text{byte}} + \mathcal{O}(\frac{T_{\text{start}} \log p}{\epsilon^2})$.

6.1.6 Ein optimaler Hyperwürfelalgorithmus

Sei $p = 2^d$. Teile Würfel in d kantendisjunkte spannende binomiale Bäume auf. Die Wurzeln der Bäume sind die Nachbarn von PE 0. Über jeden der Bäume wird $1/d$-tel der Gesamtnachricht verteilt (PE 0→Wurzel→Rest des Baums). Zeit:

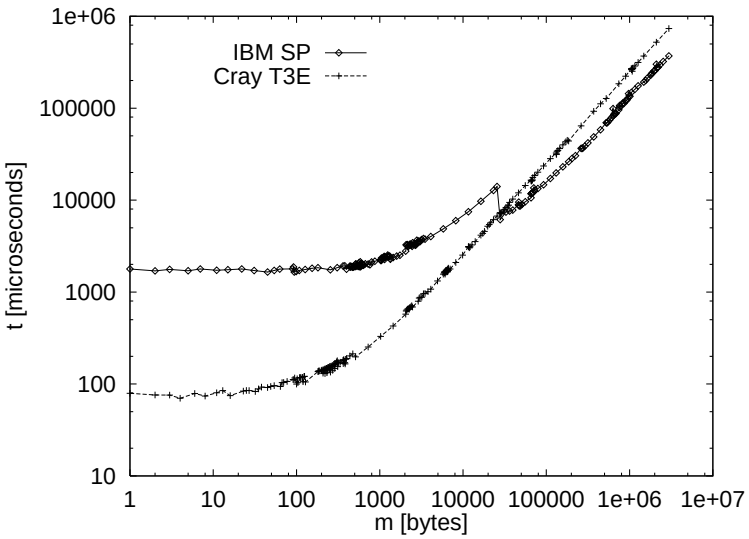
$$2nT_{\text{byte}} + \mathcal{O}(T_{\text{start}} \log p)$$

Klappt nicht auf Gittern.

6.2 Reduktion

- $\otimes$ sei assoziativer (optional zus. kommutativer) Operator.
- Berechne $\bigotimes_{i<p} x_i$, wobei x_i ein auf PE i gelegener Vektor der Länge n ist.
- Pipelined tree etc.: $T_{\text{reduce}} = \mathcal{O}(nT_{\text{byte}} + T_{\text{start}} \log p)$

6.2.1 IBM SP versus Cray T3E



6.3 Prefixsummen

[Leighton 1.2.2] Gesucht $l^{(i)} := \bigotimes_{i' \leq i} x^{(i')}$. (auf PE i, l kann ein Vektor mit n Bytes sein.)

Einfache Pipeline: Wie bei Broadcast

Pipelined Tree

Betrachte Binärbaum mit in-order numerierten Knoten.

Sei $l^{(v)} := \bigotimes_{w \in \{v\} \cup \text{leftSubTree}(v)} x^{(w)}$ (Reduktionsoperation, Zwischenergebnisse aufbewahren)

l wird in k Stücke x_j zerlegt

PE v
  for $j := 1$ **to** k **do**
    if $v \neq$ root **then**
      receive packet u from parent
      if left child exists **then** send u to left child
      $l_j := l_j \otimes u$
    if right child exists **then** send l_j to right child
  return l

Problem: Für sehr große n dreimal langsamer als einfache Pipeline.

6.4 Gossiping

Auch *All-to-all Broadcast* oder *All-Gather* genannt.

Jedes PE hat eine Nachricht x der Länge n. Am Ende soll jedes PE alle Nachrichten kennen.

6.4.1 Hyperwürfelalgorithmus

Sei $\cdot$ die Konkatenationsoperation; $p = 2^d$

PE i
  $y := x$
  for $0 \leq j < d$ **do**
    $y' :=$ the y from PE $i \oplus 2^j$
    $y := y \cdot y'$
  return y

Analyse:

$$T_{\text{gossip}}(p,n) \approx 2(p-1)nT_{\text{byte}} + 2dT_{\text{start}}$$

All-Reduce

Reduktion statt Konkatenation. Vorteil: Faktor zwei weniger Startups als Reduktion plus Broadcast. Nachteil: $p \log p$ Nachrichten. Das ist ungünstig bei stauanfälligen Netzwerken.

6.5 All-to-all Personalized Communication

Jedes PE hat $p-1$ Nachrichten der Länge n. Eine für jedes andere PE. Das lokale $x[i]$ ist für PE i

6.5.1 Hyperwürfelalgorithmus

PE i
for $j := d-1$ **downto** 0 **do**
Get from PE $i \oplus 2^j$ all its messages destined for my j-D subcube
Move to PE $i \oplus 2^j$ all my messages destined for its j-D subcube

Analyse:

$$T_{\text{all-to-all}}(p,n) \approx pndT_{\text{byte}} + 2dT_{\text{start}}$$

6.5.2 Crossbar, Multistage

Bei großem n Nachrichten lieber einzeln schicken (Faktor $\log p$ weniger Kommunikationsvolumen)

6.5.3 Ring

in, out, result : set of Msg
in := my original messages
repeat $p-1$ times
out := in
send out to right neighbor
receive in from left neighbor
move messages destined for me from in to result

Analyse:

$$T_{\text{all-to-all}}(p,n) \approx (T_{\text{start}} + npT_{\text{byte}}/2)(p-1)$$

6.6 Unregelmäßige Nachrichtenlängen

- Vor allem bei all-to-all interessant → Sortieren
- Ähnliche Probleme bei inhomogenen Verbindungsnetzwerken oder Konkurrenz durch andere Jobs.