1.11 Graph- und Schaltkreisdarstellung v. Algorithmen

Viele Berechnungen können als gerichteter azyklischer Graph dargestellt werden

- *Eingabeknoten* haben Eingangsgrad 0 und eine feste Ausgabe
- *Ausgabeknoten* haben Ausgangsgrad 0 und Eingangsgrad 1
- Der Eingangsgrad ist durch eine kleine Konstante beschränkt.
- Innere Knoten berechnen eine Funktion, die sich in konstanter Zeit berechnen läßt.

Schaltkreise

- Variante: Wenn statt Machinenworten, konstant viele bits verarbeitet werden spricht man von *Schaltkreisen*.
- Die *Tiefe* $d(S)$ des Berechnungs-DAG ist die Anzahl innerer Knoten auf dem längsten Pfad von einem Eingang zu einem Ausgang. Tiefe$\sim$Rechenzeit
- Wenn man für jede Eingabegröße (algorithmisch) einen Schaltkreis angibt, spricht man von *Schaltkreisfamilien*

Zusammenhang mit PRAMs

DAG→PRAM

Idee: Ordne den DAG S in $d(S)$ Schichten an. Berechne Operationen auf Schicht i in Phase i.

- Die Struktur muß „hinreichend einfach“ sein → Theorie *uniformer Schaltkreisfamilien* (nicht hier [z.B. Vollmar, Worsch; Modelle der Parallelverarbeitung; Teubner 1995])

PRAM→DAG

- Schleifen ausrollen
- **if then else** darf nicht von Eingabe abhängen (*oblivious computation*).

DAG → Verbindungsnetzwerke

Wir müssen den DAG in das Netzwerk *einbetten*:

- Berechnungsknoten → Prozessoren
- DAG-Kanten → **Pfade** in Verbindungsnetwerk
- Ausfuührungszeit ist Funktion der Knotenlasten, Kantenlasten und Pfadlängen in jeder einzelnen Schicht des DAG.
- Hardwarerouter: Pfadlängen sind egal.

1.12 Beispiel: Assoziative Opterationen

Satz 1. *Sei $\oplus$ ein assoziativer Operator, der in konstanter Zeit berechnet werden kann. Dann läßt sich*

$$\bigoplus_{i<n} x_i := (\cdots((x_0 \oplus x_1) \oplus x_2 \oplus \cdots \oplus x_{n-1})$$

in Zeit $\mathcal{O}(\log n)$ auf einer PRAM berechnen und in Zeit $\mathcal{O}(T_{\text{start}} \log n)$ auf einem linearen Array mit Hardwarerouter

Beispiele: $+$, $\cdot$, max, min, ... (z.B. ? nichkommutativ?)

Beweisskizze

OBdA sei $n = 2^k$. (Warum „Ohne Bedenken des Autors“?)

Wir beweisen durch vollständige Induktion über k, daß es einen Schaltkreis der Tiefe k gibt, der $\bigoplus_{i<2^k} x_i$ berechnet.

$k = 0$: trivial

$k \rightsquigarrow k+1$:

$$\bigoplus_{i<2^{k+1}} x_i = \bigoplus_{i<2^k} x_i \oplus \bigoplus_{i<2^k} x_{i+2^k}$$

Nach Induktionsannahme gibt es einen Schaltkreis für die beiden Teilsummen mit Tiefe k. Diese beiden Teilschaltkreise lassen sich durch einen $\oplus$-Knoten zu einem Schaltkreis für die Gesamtsumme der Tiefe $k+1$ zusammenfügen.

PRAM Code

PE index $i \in \{0, \ldots, n-1\}$
active := 1
for $0 \le k < \lceil \log n \rceil$ **do**
    if active **then**
        if bit k of i **then**
            active := 0
        else if $i + 2^k < n$ **then**
            $x_i := x_i \oplus x_{i+2^k}$
    (* result is in x_0 *)

Vorsicht: Viel komplizierter auf einer realen asynchronen shared memory Maschine.

Speedup? Effizienz?

$\log x$ bei uns immer $\log_2 x$

DMM Code

(Distributed Memory Machine)

PE index $i \in \{0, \ldots, n-1\}$
(* Input x_i located on PE i *)
active := 1
$s := x_i$
for $0 \le k < \lceil \log n \rceil$ **do**
    if active **then**
        if bit k of i **then**
            sync-send s to PE $i - 2^k$
            active := 0
        else if $i + 2^k < n$ **then**
            receive s' from PE $i + 2^k$
            $s := s \oplus s'$
(* result is in s on PE 0 *)

Analyse

vollst. Verk.: $\Theta((T_{\text{start}} + T_{\text{byte}}) \log n)$

lineares Array: $\Theta(n)$: Schritt k braucht Zeit 2^k.

lineares Array mit Router:
$\Theta((T_{\text{start}} + T_{\text{byte}}) \log n)$, weil edge congestion (Kantenlast) in jedem Schritt eins ist.

BSP $\Theta((l + g) \log n) = \Omega(\log^2 n)$

1.13 VLSI & Co.

- 2-D (3-D)
- Regelmäßiges Gitter aus kleinen Quadraten (Würfeln) konstanter Größe
- Komplexitätsmaße Durchmesser↔Zeit (warum?), Fläche (Volumen)↔Kosten
- These: Jedes physikalisch realisierbare (nicht quantenmechanische) Modell ist äquivalent zum VLSI Modell, z.B. scheitert beliebig genaue analoge Verarbeitung scheitert an Fertigungstoleranzen

Zellularautomaten

VLSI-Zellen→mit Nachbarn verbundene endliche Automaten.

Beispiel

Jede nichttriviale Funktion von n Eingaben braucht in einem physikalisch realisierbaren Modell Zeit $\Omega(\sqrt[3]{n})$.

- Minimum, Maximum
- Summe, ...
- Assemblerbefehl `load` einer RAM ! → Modelle für Speicherhierarchien.

1.14 Mehr zu physikalischer Realisierbarkeit

Jede noch so einfache Berechnung braucht mindestens ??s!

- Endlichkeit der Lichtgeschwindigkeit → Schranke für Zellengröße impliziert Geschwindigkeitsschranke
- Unschärferelation zwischen Energie und Zeit bedeutet Tradeoff zwischen Energieumsatz und Geschwindigkeit
- $E = mc^2$ impliziert zu Energieumsatz korrespondierende Masse
- Zu viel Masse auf zu kleinem Raum würde zu einem schwarzen Loch führen ... daraus kann keine Information entkommen

[W. J. Paul, Komplexitätstheorie, Teubner 1978]

- Die Vernichtung von einem Bit Information bei einer Temperatur von T Kelvin verbraucht mindestens $T \cdot 1.38 \cdot 10^{23}$J Energie (Boltzmannkonstante). [Feynman Lectures on Computation, 1996] $\rightarrow$ Interesse an *reversibler Berechnung*
- Dissipierte Energie muß über die Oberfläche der Maschine abfließen. $\Rightarrow$ in einer VLSI-Maschine der Größe k^3 können im Mittel nur k^2 energieverbrauchende Operationen pro Zeiteinheit stattfinden. Sonst Überhitzung. Untere Schranke! Es ist nicht trivial wie man effizient kühlt! 2D-Maschinen können die dritte Dimension zu Kühlung nutzen.

$\rightarrow$ Es gibt kein vielleicht kein universell gutes Layout für einen Compter. Sparsame Berechnungen: kugelförmig. Hoher Energieverbrauch: platt.[Sanders,Vollmar,Worsch,Europar'97]

- Quantencomputer ...

2 Arithmetik

[Leighton 1.2, (2.3)]

2.1 Addition

Eingabe: Zwei binär codierte natürliche Zahlen $a = a_n \ldots a_1$ und $b = b_n \ldots b_1$ mit n Bit.

Ausgabe: Eine binärcodierte $n+1$-Bit-Zahl $d_i \ldots d_{n+1}$.

Einfacher Schaltkreis: Kette von n Volladdierern [technische Informatik?]

Effizienter sequentieller Algorithmus: Serienaddierer

2.1.1 Carry-Lookahead-Addition

Für jeweils zwei korrespondierende Bits von a und b wird folgende $x_i = f(a_i, b_i)$ Funktion berechnet ($x_0 = s$).

$f(0,0) = s$**:** Stop. Unabhängig vom eingehenden Carrybit wird kein Carry ausgegeben

$f(0,1) = f(1,0) = p$**:** Propagate. Carry wird genau dann ausgegeben wenn von rechts eins eingegeben wird.

$f(1,1) = f(1,0) = p$**:** Propagate. Carry wird genau dann ausgegeben wenn von rechts eins eingegeben wird.

Definiere die Operation $\otimes$

```
*rspg
l\___
s|sss
p|spg
g|ggg
```

$\otimes$ ist assoziativ.

Definiere Präfixsumme $y_i := \bigotimes_{j=0}^{i} x_j$. y_i sagt vorraus ob die rechten i Bits ein Carry produzieren (warum kann kein p vorkommen?)

y_i kann durch Schaltkreis logarithmischer Tiefe berechnet werden. (Summe der Schaltkreisgrößen $\rightarrow$ später.)

Summe berechnen: $d_i = g(a_i, b_i, y_{i-1})$

	00	01	10	11
s	0	1	1	0
g	1	0	0	1

$d_{n+1} = 1$ gdw. $y_n = g$

2.1.2 Carry-Save-Addition

Eingabe: 3 binär codierte natürliche Zahlen a, a', a'' mit n Bit.

Ausgabe: Zwei binärcodierte $n+1$-Bit-Zahlen b und c

Bitweise Addition mit Carry:

$$(c_{i+1}, b_i) = a_i + a'_i + a''_i$$

2.1.3 Wallace-Tree

k Zahlen addieren (n Bit $\rightarrow n + \log k$ Bit). Reduziere mit Hilfe von Carry Save Addition in jedem Schritt um den Faktor 2/3. $\Rightarrow \approx log_{3/2}k$ Schritte bis nur noch zwei Zahlen übrig sind.

Schließlich eine Carry-Lookahead-Addition.

2.2 Multiplikation

Eingabe: n-Bit Binärzahlen a und b

- Bis zu n verschobene Kopien von b addieren; $\rightarrow$logarithmische Tiefe; $\Omega(n^2)$ Hardware.
- Verfeinerte Algorithmen in [Leighton92]
- Mit schneller Fouriertransformation sind effizientere Algorithmen für große n möglich

2.3 Division

Eingabe: n Bit natürliche Zahl y

Ausgabe: Die ersten n Bit von $1/y$

Newton Iteration: Gegeben eine differenzierbare Funktion $f(x)$ und ein x_0 'nahe genug' an einer Nullstelle. Dann konvergiert die Folge

$$x_{i+1} := x_i - \frac{f(x_i)}{f'(x_i)}$$

quadratisch gegen die Nullstelle. (D.h. in jedem Schritt verdoppelt sich die Anzahl signifikanter Bits.)

Setze $f(x) = 1 - yx$.

$f'(x) = -y$ also

$$x_{i+1} := x_i - \frac{1 - yx_i}{-y} = x_i + \frac{1}{y}(1 - yx_i)$$

Näherung: $\frac{1}{y} \approx x_i$

$$x_{i+1} = x_i + x_i(1 - yx_i) = 2x_i - yx_i^2$$

Noch zu zeigen wie schnell das konvergiert. Das klappt nur wenn $1 - 2 \leq y < 1$ ...

$o := \lceil \log y \rceil$ (* MSB finden *)
$y := y/2^o$ (* shift *)
$x := 1.5$
repeat $\log n$ times (* ausrollen *)
  $x := 2x - yx^2$
return $x/2^o$

Schaltkreistiefe $\mathcal{O}(\log^2 n)$

Weitere Tricks zur Division

- In den ersten Schritten mit weniger Genauigkeit rechnen $\Rightarrow$ Hardwareaufwand nur konstanter Faktor mehr als Multiplikation.
- Umwandlung in Repräsentation nach dem chinesischen Restesatz: Sei x eine n-Bit natürliche Zahl. Seien $p_1, \ldots\ p_s$ Primzahlen mit $P := p_1 \cdots p_s > 2^n$, dann ist der Vektor $(x \bmod p_1, \ldots, x \bmod p_s)$ eindeutig bestimmt. Mit den Resten kann man dann komponentenweise rechnen. ... diverse Tricks ... Divisionsschaltkreis mit Tiefe $\mathcal{O}(\log n)$.