Model PRAM, suma prefixů a její aplikace

Z FITwiki

Obsah

PRAM (Parallel Random Access Machine)

- synchronní model paralelního výpočtu
- **procesory se sdílenou pamětí a společným programem**
- Alternativa k paralelnímu Turingovu stroji
- Všechny procesory řízené společným programem

Procesor

- Aditivní a logické operace
- (Multiplikativní operace)
- Podmíněné skoky
- Přístup ke svému unikátnímu číslu (index)

Paměť

- Náhodný přístup pro všechny procesory
- Reprezentovaná neomezeným počtem registrů
- Neomezená délka slova (dnes není vyžadováno)
- Módy přístupu EREW, CREW a CRCW

Výpočet
: probíhá synchronně po krocích (krok: čtení, lokální operace, zápis)

Teorém - Každý problém, řešitelný PRAMem s p procesory v t krocích je také řešitelný p'≤p procesory v O(t p/p') krocích.

Architektury přístupu k paměti

- 4 možnosti
  - **EREW** - Exclusive Read, Exclusive Write (velmi omezující)
  - **CREW** - Concurrent Read, Exclusive Write (časté, jednoduché)
  - **ERCW** - Exclusive Read, Concurrent Write (nedává smysl)
  - **CRCW** - Concurrent Read, Concurrent Write (složité)
- Řešení zápisových konfliktů u CRCW
  - **COMMON** - všechny hodnoty musejí být stejné, jinak se nezapíše
  - **ARBITRARY** - zapíše se libovolná z hodnot
  - **PRIORITY** - procesory mají přidělenu prioritu, zapíše se hodnota, zapisovaná procesorem s nejvyšší prioritou

Broadcast

- **Algoritmus pro distribuci hodnoty uložené v paměti do všech procesorů**
- pro CREW a CRCW triviální v konst. čase $t(n) = O(c)$
- pro EREW je třeba simulovat současné čtení - šíření je logaritmické - jeden procesor přečte, předá dalšímu, pak jsou dva, každý jednomu, atd. $t(n) = O(log(n))$
- sekvenčně $t(n) = O(n)$

```
Algoritmus
D - hodnota, která se má rozšířit mezi N procesory
A[1..N] - pole ve sdílené paměti o délce N
procedure BROADCAST(D, N, A)
(1) A[1] = D;
(2) for i = 0 to (log N-1) do
    for j = 2^(i+1) to 2^(i+2)-1 do in parallel
        A[j] = A[j-2^i]
    endfor
endfor
```

Suma prefixů

(all-prefix-sums, allsums, **scan**)

- je základním kamenem mnoha paralelních algoritmů
- využití:
  - Vyhodnocování polynomů
  - Sčítání binárních čísel v hardware

- Lexikální porovnávání řetězců
- Implementace radix-sortu, quick-sortu

Vstup:

- uspořádaná posloupnost prvků $a_0, a_1, ..., a_{n-1}$
- binární asociativní operace $\oplus$

Výsledek:

- posloupnost $a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... \oplus a_{n-1})$

Odvozené operace

Prescan

- má na vstupu navíc neutrální prvek I
- výstup $I, a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... \oplus a_{n-2})$
- vo výstupnej postupnosti nie je posledný prvok postupnosti scan (súčet všetkých prvkov). Tento prvok je výsledkom operácie REDUCE

Reduce (paralelní suma prefixů)

- stejné vstupy jako scan
- výstupem je poslední prvek posloupnosti scan ($(a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})$)

Segmentovaný scan

- na vstupu má navíc posloupnost příznaků, které označují konce segmentů (kde je 1 tam začíná další segment)
- výstup je suma prefixů přes jednotlivé segmenty (tj. suma prefixů, která začíná od začátku v každém segmentu)

Výpočet

- optimální cena $c(n)_{opt} = t_{seq}(n)$

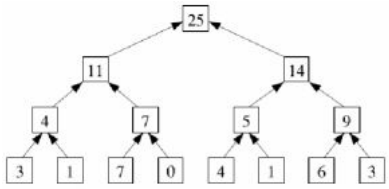
Scan - sekvenční

- t(n) = O(n)
- p(n) = 1
- c(n) = O(n)

```
procedure allsums (Out, In)
  i=0
  sum = In[0]
  while i<length do
     i = i+1
     sum = sum + In[i]
     Out[i] = sum
  endwhile
```

Reduce - paralelní (n < p)

- pomocí stromu procesorů, za předpokladu, že $\oplus$ je asociativní
- t(n) = O(log n) (strom má výšku log n)
- p(n) = n/2
- c(n) = O(n.log n) (neoptimální)




```
for j = 0 to log n - 1 do
  for i = 0 to n - 1 step 2j+1 do in parallel
    a[i+2j+1-1] = a[i + 2j -1] + a[i + 2j+1 - 1]
  end for
end for
```

Reduce - paralelní (n > p)

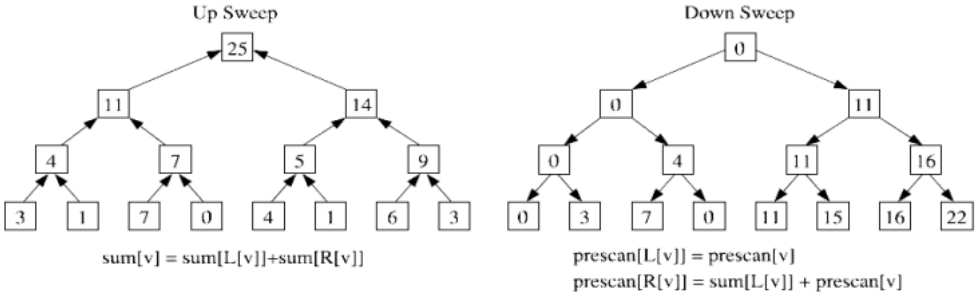
- máme méně procesorů než prvků
- každý procesor má svůj úsek, který spočítá sekvenčně, výsledky se pak spojující stromem
- pomocí stromu procesorů
- t(n) = n/N + log N = O(n/N + log N)
- c(n) = N
- c(n) = O(n/N).N = O(n) (optimální)

Prescan - paralelní

- t(n) = O(n/N)
- p(n) = N
- c(n) = O(n) (optimální za předpokladu, že log N < n/N)

Algoritmus:

- 1) **UpSweep**
  - totožné s paralelním reduce, ale každý procesor si pamatuje mezisoučet
- 2) **DownSweep**
  - kořenu se přiřadí neutrální prvek I
  - nyní se provádí log n kroků (každá úroveň jednou), počínaje kořenem, směrem k listům a v každém kroku procesory v té úrovni pracují paralelně:
  - uzel dá svému
    - P synovi svoji hodnotu ⊕ hodnotu L-syna
    - L-synovi dá svoji hodnotu



--druhy obrazek je IMO spatne, podle algoritmu je posledni radek 0 3 4 11 11 15 16 22 (viz tabulka PRL - H006 str. 20)

--druhy obrazek je ZCELA URCITE SPATNE, Hanacek ma ve slajdech chybu, je to tak jak pise kolega nade mnou (by Conflict)

Scan - paralení

- získáme z prescan tak, že výsledky posuneme doleva a zprava doplníme výsledkem Reduce.

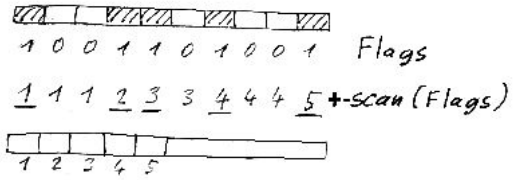
Použití

Článok (http://www.google.com/url?sa=t&source=web&cd=1&ved=0CBUQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.128.6230%26rep%3Drep1%26type%3D
a-wa41eCEBw&usg=AFQjCNEXx7rqJz77cmxTkQohGWfnW0Or4A&sig2=2tFryidoPi932OjZFGc2eQ&cad=rja) z ktorého čerpal Hanáček

- vyhodnocování polynomů
- rychlé sčítání v HW
- lexikální analýza a porovnávání
- řazení
- hledání regulárních výrazů
- odstranění označených prvků
- apod.

Packing problem

- **Problém:** máme pole v němž jsou náhodně rozmístěny prvky, potřebujeme pole ve kterém budou prvky na jeho začátku v původním pořadí
- **Postup:**
  - 1) Vytvoříme pole příznaků (1 na pozicích, kde je ve vst. poli prvek)
  - 2) Provedeme +-scan na pole příznaků
  - 3) Každý prvek přesuneme na pozici, která je u něj v poli příznaků

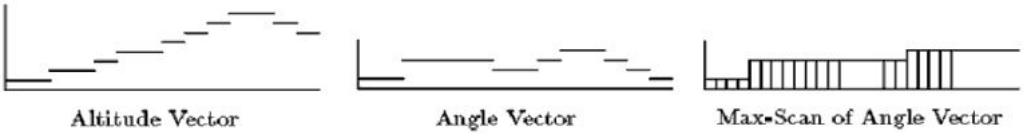


Viditelnost

- **Problém:** máme vektor výšek terénu podél sledovacího paprsku. Zjistit, které body terénu jsou viditelné
- **Řešení:**
  - Bod je viditelný, pokud žádný bod mezi pozorovatelem a jím nemá větší vertikální úhel.

- 1) vytvoří se vektor výšek bodů podél pozorovacího paprsku
- 2) vektor výšek se přepočítá na vektor úhlů
- 3) pomocí max_prescan se spočte vektor maximálních úhlů pro zjištění viditelnosti bodu stačí určit jeho úhel a porovnat s maximem.

- t(n) = O(n/N + log N) na EREW PRAM



Radix sort

- **Problém:** Radix sort
  - V každém kroku se pomocí operace split prvky s n-tým bitem 0 přemístí na začátek řazených čísel a s n-tým bitem 1 na konec
- **Postup:**
  - pro prvky spočítáme jejich pozice a pak v konstantním čase přemístíme
  - pro prvky s nulovým bitem se jejich pozice získá provedením ⊕ - prescan na invertované pole bitů
  - pro prvky s jedničkovým bitem provedu ⊕ scan na reverzované pole bitů (tj. od konce) a výsledek se odečte od n.
- t(n) = O(n/N + log N).O(log n) = O(n/N.log n + log n . log N)

```
procedure split(A, Flags)
  I-down = +-prescan(not(Flags))
  I-up = n - +-scan(reverse-order(Flags))
  for i=0 to n-1 do in parallel
        if (Flags[i]) Index[i] = I-up[i]
        else Index[i] = I-down[i] endif
  endfor
  result = permute (A, Index)
```

Quicksort

- **Problém:** Quicksort
  - Jeden z prvků se vybere jako pivot (medián, náhodně, první)
  - prvky se rozdělí do 3 skupin (menší, rovné, větší než pivot)
  - pro každou skupinu se rekurzivně volá quicksort
- **Postup:**
  - Použije se segmentovaný scan a každá skupina bude ve svém vlastním segmentu
  - 1) zkontroluj, zda už prvky nejsou seřazené
    - Každý procesor se podívá, zda předchozí procesor má menší, nebo stejnou hodnotu.
    - S výsledky se provede and-reduce
  - 2) v každém segmentu najdi pivot a předej jej ostatním procesorům v segmentu.
    - Vybírá-li se jako pivot 1. prvek, lze použít segmented-copy-scan, kde binární operátor copy vrací 1. ze svých 2 parametrů: a ← copy(a, b)
    - (lze také pivota vybírat jinak)
  - 3) v každém segmentu porovnej prvky s pivotem a rozděl segment na 3 části
    - Pro rozdělení se použije modifikovaný split z radix-sortu.
  - 4) v rámci každého segmentu vlož dodatečné příznaky, které rozdělí segment na 3 segmenty.
    - Každý procesor se podívá na předchozí prvek a pozná, zda je na začátku segmentu.
  - 5) jdi na krok 1)
- Každá iterace má konstantní počet operací scan
- Při vhodné volbě pivotů skončí algoritmus po O(log n) krocích,
- t(n) = O(n/N + log N).O(log n) = O(n/N.log n + log N. log n)
- c(n) = O(n.log N + N. log n. log N) (pro dostatečně malé N optimální)

Kategorie: Státnice 2011 | Paralelní a distribuované algoritmy

---