Approfondimento Algoritmica

Liva Giovanni

Università di Udine

25 agosto 2014

Argomenti

Introduzione

Vari modelli di calcolo, si differenziano sul livello di astrazione dal modello reale

- *VLSI*: Attenzione ai limiti fisici dei processori
- *PRAM*: Modello teorico non implementabile che non considera i problemi di comunicazione
- *Circuiti*: Lo stesso modello visto a lezione può essere usato per lo studio di algoritmi paralleli

Principio di Brent

Abbiamo una computazione parallela eseguita in t passi.

- Supponiamo di avere x_i operazioni ad ogni passo i
- Il numero di processori necessario è $d = max\ x_i$
- Possediamo solo $p < d$ processori
- Possiamo simulare la computazione originaria con p processori in $\lceil x_i/p \rceil$ passi ad ogni i-esimo step
- In totale la simulazione viene eseguita in $\lceil \frac{\sum_i x_i}{p} \rceil + t$ passi

Parallel Random-Access Machine

- Si basa sul modello delle *RAM* introducendo una memoria globale (*registro accumulatore*) dove le varie *RAM* possono comunicare
- In tempo $\mathcal{O}(1)$ possiamo r/w una cella della memoria locale o globale oppure eseguire una operazione *RAM*

Un programma *PRAM* $\mathbb{P} = (\Pi_1, \ldots, \Pi_q)$ è fatto da q macchine *RAM* indipendenti dove q è una funzione $q(m, n)$ dove $m = |I|$ ed $n = \ell(I)$. Normalmente il numero di *RAM* richiesto dipende solo da m.

In base a come gestiamo i conflitti di r/w sul registro accumulatore abbiamo 3 diverse tipologie di PRAM

- Exclusive-Read Exclusive-Write (*EREW*)
- Concurrent-Read Exclusive-Write (*CREW*)
- Concurrent-Read Concurrent-Write (*CRCW*)

Per risolvere i conflitti di scrittura abbiamo 3 metodi:

- *Common*: Tutti i processi che insistono in una stessa locazione devono scrivere lo stesso valore
- *Arbitrary*: Tra tutti i processi che provano a scrivere, solo uno ha successo. L'algoritmo deve comunque funzionare a prescindere da chi vince
- *Priority*: Il processo l'identificativo più basso è quello che scrive

Parallel Random-Access Machine :: Tipologie e Ugualianze

L'ordine con il quale sono state presentate è anche l'ordine di potenza dei vari modelli.
Tra loro però sono correlati da un fattore logaritmico. Per cui il modello più potente, *CRCW Priority* può essere simulato da una *EREW* con lo stesso numero di processori e con un tempo parallelo aumentato di $\mathcal{O}(log\ P)$; dove P è il numero di processori.

- Dimostrazione a voce

Parallel Random-Access Machine :: Classe *NC*

Possiamo definire una classe per gli algoritmi paralleli chiamata NC dove:

$$NC = \bigcup_{k>0} CRCW(n^k, \log^k n)) \tag{1}$$

e dal fatto che i vari modelli di $PRAM$ sono correlati da un fattore logaritmico:

$$NC = \bigcup_{k>0} PRAM(n^k, \log^k n)) \tag{2}$$

Vengono anche definite sotto-classi di NC nel seguente modo:

$$NC_j = \bigcup_{k>0} PRAM(n^k, \log^j n)) \tag{3}$$

Rimane ancora aperto il problema $NC \stackrel{?}{=} P$

Parallel Random-Access Machine :: Algoritmo Ottimale

Definizione

$polylog(n) = \bigcup_{k>0} \mathcal{O}(log^k n)$
Sia S un programma sequenziale che opera in tempo $T(n)$.

Diremo che il programma A di una *PRAM* per S che opera in tempo $t(n)$ con $p(n)$ processori è ottimale se:

- $t(n) = polylog(n)$
- $w(n) = p(n) \times t(n) = \mathcal{O}(T(n))$

Identifichiamo il lavoro eseguito da una *PRAM* con la coppia $(w(t), t(n))$

Circuiti :: Definizioni

Definizione

Un circuito è un grafo etichettato aciclico (DAG). Le etichette dei nodi possono essere input, costanti, AND, OR, NOT oppure output.
Input e costanti hanno 0 archi in ingresso, output e NOT 1, mentre AND e OR 2. I nodi di output hanno 0 archi in uscita.

Definizione

Sia $\mathbb{B} = \{0, 1\}$. Un circuito con n input e m output calcola la funzione $f : \mathbb{B}^n \to \mathbb{B}^m$. Si suppone che l'input sia ordinato e così l'output.

Definizione

La *profondità* di un circuito è la lunghezza del più lungo cammino da un nodo di input ad uno di output.
La *dimensione* di un circuito è il numero di nodi che lo compongono

Circuiti :: Definizioni

Definizione

Una famiglia di circuiti $C = \{C_i\}, i = 1, 2, \ldots$ dove ogni C_i ha esattamente i input e $\ell(i)$ output.
La famiglia C di circuiti risolve un problema P se la funzione compiuta da C_i produce la stessa trasduzione di P sulla stringa di input lunga i

Definizione

Una famiglia di circuiti C è *logspace uniforme* se il circuito C_n può essere generato da una macchina di Turing in spazio $\mathcal{O}(\log n)$

Definizione classe CKT

Per $C(n) \geq n$ e $D(n) \geq n$ indichiamo con $CKT(C(n), D(n))$ la classe dei circuiti C *logspace uniforme* che risolve il problema P con dimensione $\mathcal{O}(C(n))$ e profondità $\mathcal{O}(D(n))$

Circuiti :: Unbounded

Definizione

Unbounded fan – in circuit: rimuoviamo la restrizione sul numero di input per i nodi AND e OR
Hanno una correlazione logaritmica con i cricuiti classici, possiamo convertire ogni nodo con un albero fatto da nodi con 2 input

- Come per i circuiti classici possiamo dare la definizione della Classe UCKT
- Se $P \in UCKT(p(n), d(n)) \Longrightarrow P \in CKT(p(n), d(n) \times \log p(n))$

Circuiti :: Relazioni con *PRAM*

Dato un circuito $C \in UCKT(S(n), D(n))$ lo possiamo simulare con una *CRCW* in tempo $\mathcal{O}(D(n))$ e con $S(n)$ processori usando $n + M(n)$ locazioni di memoria,dove $M(n)$ è il numero di *gates*.

Dimostrazione

- Portiamo i nodi NOT sugli input (cheat)
- I processori corrispondono agli archi del grafo, le locazioni di memoria ai nodi
- L'input è dalla locazione 1 alla n, le locazioni $n + i$ con $i \in \{1 \ldots M(n)\}$ sono inizializzate con 0 se OR e 1 se AND
- Ad ogni passo, il processore p determina il valore corrente c sull'arco $e = (u, v)$ leggendo la locazione $n + u$
- Se $c = 0 \wedge v = \text{OR}$, oppure $c = 1 \wedge v = \text{AND} \Longrightarrow$ scrive il valore di c nella locazione $n + v$ eseguendo la relativa operazione
- Se $c = 1 \wedge v = \text{OR}$, oppure $c = 0 \wedge v = \text{AND} \Longrightarrow$ scrive il valore di c nella locazione $n + v$

Circuiti :: Relazioni con *PRAM*

La dimostrazione inversa invece prevede di fornire un circuito per ogni operazione di una RAM. I circuiti, per come si è potuto vedere durante il laboratorio di programmazione, sono tutti a profondità costante. Ne consegue quindi:

$$CRCW(P(n), T(n)) \subseteq UCKT(poly(P(n)), T(n)) \tag{4}$$

Dove $poly(n) = \bigcup_{k>0} n^k$

Macchine di Turing Alternate

- Derivano dalla teoria degli automi alternati
- Sono una generalizzazione delle MdT non deterministiche
- Ogni stato è esistenzione oppure universale

A partire da uno configurazione α possiamo dire che è di accetazione se:

- Se α è **esistenziale** allora esiste una computazione che porta ad una configurazione *yes*
- Se α è **universale** allora tutte le computazioni a partire da α arrivano ad una configurazione *yes*

Si dimostra che: $ATM(S(n), T(n)) \subseteq CKT(c^{S(n)}, T(n))$ con $ATM(S(n), T(n))$ la classe delle macchine di Turing alternate che operano in spazio $\mathcal{O}(S(n))$ e tempo $\mathcal{O}(T(n))$.
Un ulteriore risultato è: $NC = ATM(\log n, polylog(n))$

Problemi :: Matrix Multiplication

- Siano A e B due matrici $n \times n$, dobbiamo calcolare $C = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}, \quad i, j = 1, \ldots, n$
- Sequenzialmente il problema può essere risolto in $\mathcal{O}(n^3)$
- Usiamo n^3 processori; ogn'uno può essere identificato da una tripla (i, k, j)
- Organizziamo in un albero le varie somme

Eseguiamo un primo passo dove ogni processore svolge la sua moltiplicazione e poi eseguiamo le $\log n$ somme.
Al passo s il processore $(i, 2^s \lfloor \frac{k}{2^s} \rfloor, j)$ computa la somma della sua moltiplicazione con quella del processore $(i, 2^s \lfloor \frac{k}{2^s} + 2^s \rfloor, j)$. In totale abbiamo $\lceil \log n \rceil - 1$ somme e il risultato si troverà sui processori $(i, 1, j)$

Problemi :: Reachability

Il problema in un primo approccio non sembra parallelizzabile, infatti per riuscirci bisogna abbandonare il classico metodo con il quale ci si approccia al problema.
Consideriamo la matrice di adiacenza A come una matrice booleana, dove mettiamo un selfloop su tutti i nodi: $A_{i,i} = 1$
Computando ora $A^2 = A \cdot A$ abbiamo che $A^2_{i,j} = \bigvee_{k=1}^{n} A_{i,k} \wedge A_{k,j}$, o meglio $A_{i,j} = 1$ sse esiste un percorso di lunghezza al massimo 2 da i a j.

Problemi :: Reachability

Se computiamo per $\lceil \log n \rceil$ il prodotto, otteniamo la matrice $A^{2^{\lceil \log n \rceil}}$ che corrisponde alla chiusura transitiva di A che corrisponde alla risposta di tutte le possibili istanze di Reachability.
In conclusione:

- $\log n$ moltiplicazioni e quindi $\mathcal{O}(\log^2 n)$ lavoro per processore
- n^3 processori neccessari, dovuto a *Matrix Multiplication*

Problemi :: Prefix Sum

Dato un array di n elementi $x = [x_1, \ldots, x_n]$ si vuole calcolare $\sum_{i=1}^{j} x_i$ per $j = 2, \ldots, n$

Per computare le varie somme con un algoritmo parallelo, come prima cosa eseguiamo le seguenti somme: $(x_1 + x_2), \ldots, (x_{n-1} + x_n)$.

In questo modo otteniamo un nuovo vettore y fatto da $\lceil \frac{n}{2} \rceil$ elementi.

Possiamo riapplicare il ragionamento fino ad ottenere un unico elemento, in questo modo otteniamo la somma di tutti gli elementi in posizione pari.

Ora, per computare le dispari eseguiamo un primo passo di somme: $x_1, (x_2 + x_3), \ldots$

Al nuovo vettore appena ricavato applichiamo l'algoritmo visto prima.

L'algoritmo ha la seguente equazione ricorsiva:

$$T(n) = T\left(\frac{n}{2}\right) + \mathcal{O}(1) \tag{5}$$

Di conseguenza l'algoritmo impiega tempo $\mathcal{O}(\log n)$ e n processori.

Fine