PARALLEL HIGH PERFORMANCE COMPUTING

CdS magistrale in informatica

5 – prodotto di matrici su CPU multicore

Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

wpage.unina.it/lapegna

1

prodotto di matrici C = C + A*B

numerosi problemi di algebra lineare di base (somma di matrici, prodotto tra una matrice ed un vettore) sono naturalmente **predisposti al calcolo parallelo** in quanto è spesso possibile operare concorrentemente sui dati senza utilizzare sezioni critiche.

Ciò vale anche per il prodotto di matrici

C = C + A*B

dove A e' una matrice di NxM elementi, B di MxP elementi e di NxP elementi.

Al fine di valutare i vari algoritmi su sistemi paralleli a memoria condivisa è allora opportuno **fare riferimento alla Legge di Amdahl**

$$S_P = \frac{P}{(1 + \frac{O_h(P)}{T(N,1)})}$$

che mostra l'influenza dell'overhead

$$O_h(P) = P\,T(N,P) - T(N,1)$$

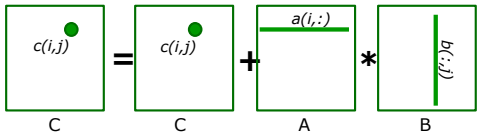
sullo Speed-up e sull'Efficienza

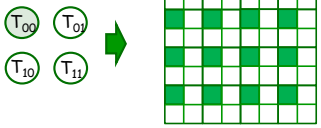
2

algoritmo 1

un primo algoritmo può essere ottenuto osservando che ciascun elemento C(i,j) può essere calcolato, **indipendentemente dagli altri**, come il **prodotto scalare** di una **riga di A e una colonna di B**

$$c(i,j) = \sum_{k=0}^{M-1} a(i,k)b(k,j) \qquad i = 0,..,N-1 \qquad j = 0 \ldots, P-1$$





Utilizzando NT thread, organizzati come una griglia di NTrow x NTcol unita' di calcolo, l'Algoritmo 1 prevede che ciascuna di esse calcoli, ciclicamente sulle righe e sulle colonne, N^2/NT elementi della matrice C.

3

analisi dell'overhead

Per calcolare l'overhead dell'Algoritmo 1 si studi, per semplicità, il caso di matrici quadrate tutte di ordine N, e si indichi con P=NT=NTrow x NTcol il numero di thread.

Per P=1 thread è necessario accedere alle intere matrici A e B in lettura e alla matrice C in lettura e scrittura. Ricordando che il numero di operazioni è $2N^3$, si ha che il tempo per l'esecuzione è

$$T(N,1) = 4N^2\,t_{mem} + 2N^3 t_{flop}$$

Nel caso P> 1, per calcolare un singolo elemento della matrice C, ciascun thread deve accedere ad una riga di A e a una colonna di B per complessivi $2N+2$ accessi alla memoria (il thread accede anche a un singolo elemento di C in lettura e scrittura), e deve eseguire $2N$ operazioni. Poiché ogni thread deve calcolare complessivamente N^2/P elementi della matrice C, si ha che il tempo di esecuzione con P thread è:

$$T(N,P) = \frac{N^2}{P}\left((2N+2)t_{mem} + 2N t_{flop}\right)$$

L'overhead è allora:

$$O_h(P) = P\,T(N,P) - T(N,1) = P\frac{N^2}{P}\left((2N+2)t_{mem} + 2N t_{flop}\right) - \left(4N^2\,t_{mem} + 2N^3 t_{flop}\right) = (2N^3 - 2N^2)t_{mem}$$

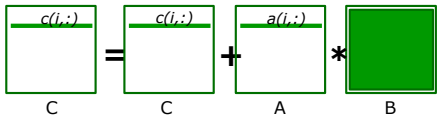
Anche in assenza di sezioni critiche, la parallelizzazione del problema ha introdotto un **overhead che dipende dall'accesso ai dati.** Ciò è dovuto al fatto che i thread devono accedere più volte, indipendentemente tra loro, agli stessi elementi delle matrici A e B per calcolare i valori di C di propria competenza.

4

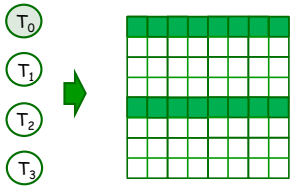
algoritmo 2

Un secondo algoritmo può essere ottenuto osservando che ciascuna riga c(i,:) della matrice C può essere calcolata, **indipendentemente dalle altre**, come il prodotto tra la **corrispondente riga a(i,:) della matrice A e l'intera matrice B**

$$\boldsymbol{c(i,:) = a(i,:)\,B} \qquad \boldsymbol{i = 0,..,N-1}$$



Utilizzando NT thread disposti in maniera ordinata secondo una sequenza lineare, ciascuno di essi ha allora il compito di calcolare **ciclicamente** N/NT righe di C.



analisi dell'overhead

per calcolare l'overhead si studi, per semplicità, il caso di matrici quadrate di ordine N, e si indichi con P=NT il numero di thread. A tal fine, si osservi che per calcolare il risultato con P=1 thread è necessario accedere alle intere matrici A e B in lettura e alla matrice C in lettura e scrittura. Il tempo per l'esecuzione è quindi:

$$T(N,1) = 4N^2\, t_{mem} + 2N^3 t_{flop}$$

Nel caso P>1 thread, per calcolare una riga di C ogni thread deve accedere, indipendentemente dagli altri, a una riga di A e all'intera matrice B in lettura, e ad una riga di C in lettura e scrittura, per complessivi N^2+3N accessi alla memoria. Deve inoltre eseguire $2N^2$ operazioni.

Poiché ciascuno di essi deve calcolare N/P righe di C, il tempo complessivo di esecuzione è:

$$T(N,P) = \frac{N}{P}\left((N^2+3N)t_{mem} + 2N^2 t_{flop}\right)$$

L'overhead è allora:

$$O_h(P) = P\,T(N,P) - T(N,1) = P\frac{N}{P}\left((N^2+3N)t_{mem} + 2N^2 t_{flop}\right) - \left(4N^2\, t_{mem} + 2N^3 t_{flop}\right) = N^3 - N^2)t_{mem}$$

Tale overhead è inferiore (circa la metà) di quello ottenuto per l'Algoritmo 1

Nei due algoritmi presentati, si è osservato che **l'overhead dipende essenzialmente dal numero di accessi alla memoria**. In entrambi i casi il numero di accessi alla memoria è **dell'ordine di N^3** a fronte di un numero di operazioni che è sempre **$2N^3$**

E' importante osservare ora che, disponendo di cache memory sufficientemente grandi da poter contenere tutti i dati del problema, si ha che per eseguire il prodotto di matrici sarebbero necessari solo **$4N^2$** accessi. Per tale motivo, **i due algoritmi studiati introducono un overhead dell'ordine di N^3 che è da considerarsi troppo elevato per ottenere dei ragionevoli valori per lo Speed-up.**

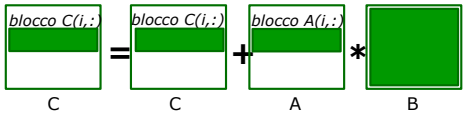
Tale osservazione suggerisce anche che, suddividendo **la matrice C in blocchi** (da assegnare a thread distinti), e calcolando questi come **prodotto tra matrici effettuato tra blocchi della matrice A e blocchi della matrice B,** si può beneficiare del migliore rapporto tra numero di accessi alla memoria e numero di operazioni eseguite caratteristico di tale operazione matriciale.

Si osservi infine che l'ipotesi sulla dimensione della cache memory è da considerarsi non restrittiva, in quanto si è visto come superare la limitazione attraverso lo sviluppo di algoritmi a blocchi.

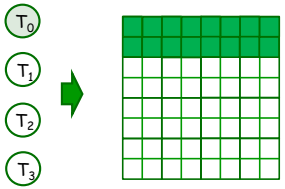
algoritmo 3

Si dividano allora la matrici A e C in blocchi di N/NT righe. Un terzo algoritmo è allora ottenuto osservando che ciascun blocco C(i,:) di righe di C può essere calcolato, **indipendentemente dagli altri**, come il prodotto scalare tra il **corrispondente blocco di righe A(i,:) di A e la matrice B**

$$\boldsymbol{C(i,:) = A(i,:)\,B} \qquad \boldsymbol{i = 0,..,NT-1}$$



Utilizzando NT thread disposti in maniera ordinata secondo una sequenza lineare, ciascuno di essi ha allora il compito di calcolare un blocco di N/NT righe di C come un prodotto tra matrici.



analisi dell'overhead

Nel caso P>1 ciascun thread deve accedere, indipendentemente dagli altri, ad un blocco di A e all'intera matrice B in lettura, e ad un blocco di C in lettura e scrittura. Osservando che i blocchi delle matrici A e C hanno N/P righe e N colonne, e che il numero di operazioni è $2N^3/P$ si ha che il tempo di esecuzione di ciascuno di essi è:

$$T(N,P) = \left(N^2 + 3\frac{N^2}{P}\right)t_{mem} + 2\frac{N^3}{P}t_{flop}$$

L'overhead totale è allora:

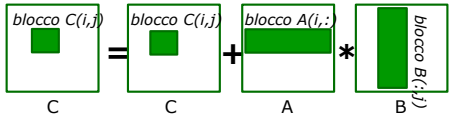
$$O_h(P) = P\,T(N,P) - T(N,1) = P\left(\left(N^2 + 3\frac{N^2}{P}\right)t_{mem} + 2\frac{N^3}{P}t_{flop}\right) - \left(4N^2\,t_{mem} + 2N^3 t_{flop}\right) = (P-1)N^2 t_{mem}$$

Poiché di solito N>>P, è facile verificare che tale overhead è molto inferiore di quello ottenuto per l'Algoritmo 1 e l'Algoritmo 2. Tale miglioramento è stato ottenuto grazie al minor numero di accessi alla memoria rispetto al numero di operazioni del prodotto di matrici eseguito da ciascun thread, rispetto ai nuclei computazionali dei precedenti due algoritmi.

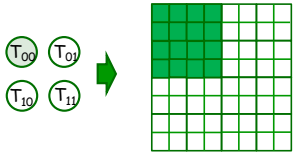
algoritmo 4

Un ultimo algoritmo può essere ottenuto utilizzando NT thread disposti secondo una griglia di dimensioni NTrow x NTcol, e suddividendo la matrice C in **blocchi di dimensione** N/NTrow righe e P/NTcol colonne. In questo caso ciascun blocco **C(i,j)** della matrice C può essere calcolato, **indipendentemente dagli altri**, come il prodotto tra un **blocco A(i,:) di N/NTrow righe e M colonne di A e un blocco B(:,j) di M righe e P/NTcol colonne di B**

*C(i,j) = A(i,:) * B(:,j) i= 0,…, NTrow-1 j=0,…, NTcol-1*



Disponendo i NT thread in maniera analoga ai blocchi della matrice C, l'Algoritmo 4 prevede che ciascuno di essi calcoli un blocco di N/NTrow righe e P/NTcol colonne della matrice C.



analisi dell'overhead

si studi, per semplicità, il caso di **matrici quadrate tutte di ordine N,** e si indichi con P=NT=NTrow x NTcol il numero di thread.

In questo caso ogni thread deve accedere, indipendentemente dagli altri,

- al blocco C(i,j) che richiede N^2/P accessi in lettura e in scrittura
- al blocco A(i,:) che richiede $N^2/NTrow$ accessi in lettura
- al blocco B(:,j) che richiede $N^2/NTcol$ accessi in lettura

Si ha che il tempo di esecuzione di ogni thread è:

$$T(N,P) = \left(2\frac{N^2}{P} + \frac{N^2}{NTrow} + \frac{N^2}{NTcol}\right)t_{mem} + 2\frac{N^3}{P}t_{flop}$$

L'overhead totale è allora:

$$O_h(P) = P\,T(N,P) - T(N,1) = P\left(\left(2\frac{N^2}{P} + \frac{N^2}{NTrow} + \frac{N^2}{NTcol}\right)t_{mem} + 2\frac{N^3}{P}t_{flop}\right) - \left(4N^2\,t_{mem} + 2N^3 t_{flop}\right)$$

da cui si ricava:

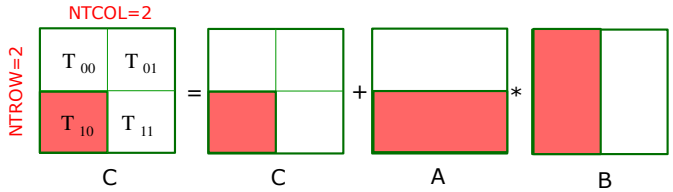
$$O_h(P) = (NTrow + NTcol\ - 2)N^2 t_{mem}$$

L'overhead ottenuto è ancora inferiore rispetto a quello dell'Algoritmo 3.

E' interessante osservare che l'Algoritmo 3 rappresenta un caso particolare dell'algoritmo 4 nel caso NTrow = P e NTcol = 1

elaborato: prodotto matrici multithread

- dividere la matrice C in blocchi pari al numero di thread,
- identificare i thread con una coppia di indici (cosi' da rispettare naturalmente la suddivisione della matrice in blocchi)
- Esempio: NTROW = 2, NTCOL = 2 (NT = 4 thread)



- ogni thread esegue il calcolo di un blocco della matrice C

elaborato: prodotto matrici multithread

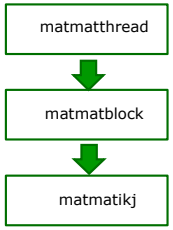
testata della function

```
void matmatthread (int lda, int ldb, int ldc,
                   float *A, float *B, float *C,      int N,  int M,  int P,
                   int dbN,   int dbM,   int dbP,     int ntrow,  int ntcol)
```

riutilizzare la function matmatblock (dbN = 256)

esperimenti con matrici di ordine
N=M=P=512, 1024, 1536, 2048

misurare speedup e efficienza con 1, 2 e 4 thread



....
id = omp_get_....
idrow = id/NTCOL
idcol = id % NTCOL
matmatblock(LDA, LDB, LDC, A(idrow*N/NTROW , 0)
B (0 , idcol*P/NTCOL) , C(idrow*N/NTROW, idcol*P/NTCOL),
N/NTROW, M , P/NTCOL , dbN, dbM, dbP)