Dipartimento di Matematica e Informatica

Laurea Magistrale in Informatica

Corso di

Programmazione Concorrente e Parallela \ Metodi teorici e computazionali per le scienze molecolari

Implementazione parallela del Metodo di Jacobi per il calcolo di autovalori e autovettori

studenti
Luca Pagliochini
Matteo Riganelli

docenti
Leonardo Pacifici
Antonio Laganà
Sergio Tasso

2015/2016

Introduzione

- Algoritmo matematico
- Parallelizzare algoritmo
- Speed Up del metodo utilizzato

L’algoritmo di Jacobi

- Carl Gustav Jacob Jacobi (1846) ma ampiamente utilizzato dal 1950 (avvento computer)
- Trasformazione Matrice simmetrica
- Annichilire elementi al di sotto della diagonale della matrice data
- operazioni eseguite: *Jacobi Rotation*
- Restituisce in output:
    - autovalori
    - autovettori

```
ROTATE(a,i,j,k,l)
g=a[i][j];
h=a[k][l];
a[k][l]=h+s*(g-h*tau);
a[i][j]=g-s*(h+g*tau);
```

L’algoritmo di Jacobi

```
for (ip=1; ip<=n-1; ip++)  (1°for)
    for (iq=ip+1; iq<=n; iq++) (2°for)
        for (j=1, j<=ip-1)    ROTATE(a,j,ip,j,iq)
        for (j=ip+1 j<=iq-1) ROTATE(a,ip,j,j,iq)
        for (j = iq+1 < n)    ROTATE(a,ip,j,iq,j)
        for (j =1, j<=n)      ROTATE(v,j,ip,j,iq)
```

Algoritmo Seriale - dove parallelizzare ?

```
int n=6;
(1°iterazione)
for ip=1; ip<=n-1; ip++  (1°for)
     for iq=ip+1; iq<=n; iq++    (2°for)
                  ip=1; iq=2
for j=1 a ip-1     ROTATE(a,j,ip,j,iq)
for j=ip+1 a iq-1 ROTATE(a,ip,j,j,iq)
for j = iq+1 a n  ROTATE(a,ip,j,iq,j)
           a[1][3], a[2][3],
           a[1][4], a[2][4],
           a[1][5], a[2][5],
           a[1][6], a[2][6]
for j =1 a n  ROTATE(v,j,ip,j,iq)
           v[1][1], v[1][2],
           v[2][1], v[2][2],
           v[3][1], v[3][2],
           v[4][1], v[4][2],
           v[5][1], v[5][2],
           v[6][1], v[6][2]
```

```
(2°iterazione)
for ip=1; ip<=n-1; ip++  (1°for)
     for iq=ip+1; iq<=n; iq++  (2°for)
                  ip=1; iq=3
for j=1 a ip-1     ROTATE(a,j,ip,j,iq)
for j=ip+1 a iq-1 ROTATE(a,ip,j,j,iq)
           a[1][2], a[2][3]
                    ...
```

Algoritmo Parallelo

- 1° versione:
  - invia ogni singola operazione di rotate allo slave e si mette in attesa di ricevere il risultato per aggiornare la matrice
  - vengono parallelizzati i cicli for che si occupano di effettuare le operazioni di rotate
  - matrice A e V
  - dinamico
  - oneroso
- 2° versione
  - come la versione 1 però applicato solo a livello dell'ultimo ciclo for (ultima Rotate)
  - matrice V
  - oneroso

Algoritmo Parallelo

- 3° versione:
  - miglioramenti della versione 2
  - calcolo degli slave basato su un range di indici (chunk)
  - molto più performante
- 4° versione
  - come versione 3, ma utilizzo della primitiva MPI_Bcast()
  - migliora i tempi di invio dei dati agli slave

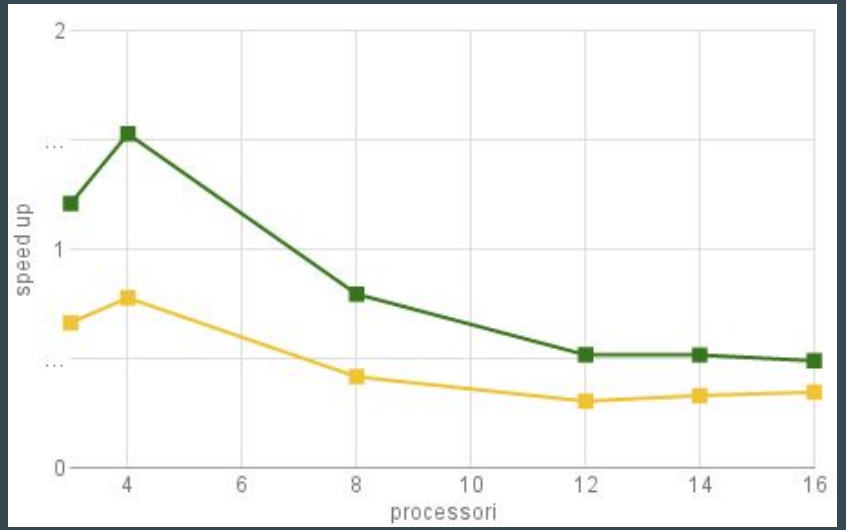
Algoritmo Seriale

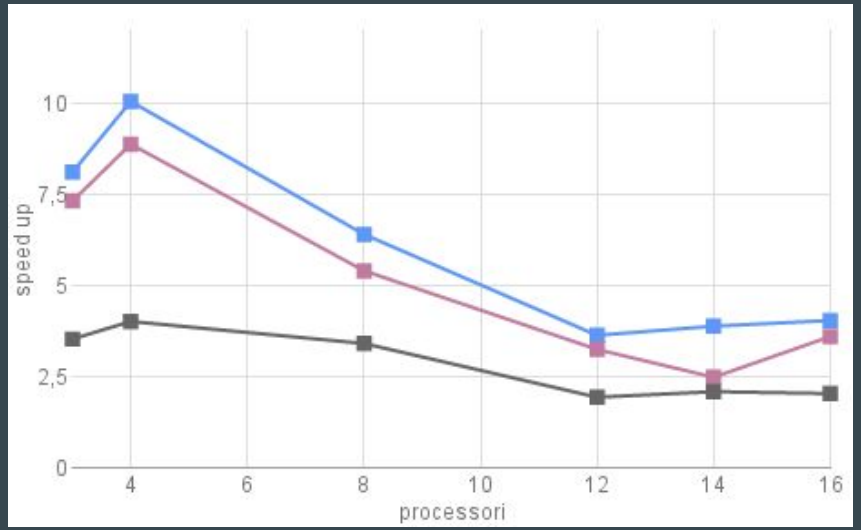
Matrice	Tempo Esecuzione (s)
500	14,203533
700	58,5994
1000	1233,367891
1200	2116,1571
1700	5264,408303

Algoritmo Parallelo - versione 4

Matrice	Processori	Tempo Esecuzione (s)
500	4	18,2885
700	4	38,37855
1000	4	122,7225
1200	4	238,38185
1700	4	1311,47025

Algoritmo Parallelo - Speed Up





500 x 500	700 x 700	1000 x 1000	1200 x 1200	1700 x 1700

Conclusioni

- performance buone algoritmo seriale
- Algoritmo seriale non effettua operazioni costose
- limitazione nei punti dove parallelizzare
- soluzione adottata: range dinamici

Grazie per l’attenzione