Parallelizzazione con MPI del Metodo di Jacobi

Progetto APSD 2017

Laurito Domenico 176220

Il Metodo di Jacobi

- il metodo di Jacobi è un metodo iterativo per la risoluzione di sistemi lineari, un metodo cioè che calcola la soluzione di un sistema di equazioni lineari dopo un numero teoricamente infinito di passi. Per calcolare tale risultato il metodo utilizza una successione che converge verso la soluzione esatta del sistema lineare e ne calcola progressivamente i valori arrestandosi quando la soluzione ottenuta è sufficientemente vicina a quella esatta. Fu ideato dal matematico tedesco Carl Jacobi.

Il Metodo di Jacobi

- Il metodo Jacobiano di risoluzione è considerato uno dei più semplici tra quelli iterativi Il metodo di Jacobi è il più semplice tra tutti quelli iterativi, che mirano a risolvere sistemi Di equazioni lineari Ax=b.
- Questo porta il metodo ad avere anche aspetti negativi, infatti spesso non viene usato data la sua lentezza di esecuzione, essendo appunto iterativo.
- Tuttavia esso viene riconsiderato con l'avvento della programmazione parallela.

Metodo Jacobiano

- $a^{(k)} = (x1^{(k)}, x2^{(k)}, x3^{(k)}, \dots, xn^{(k)})$
- Per trovare il valore della x, la strategia del metodo di Jacobi è quella di scegliere inizialmente un valore per le incognite(es. zero), successivamente risolvendo l'equazione il valore trovato va a sostituire le incognite della equazione ecc… Questo avviene fino a che il risultato trovato non rispetta la tolleranza impostata(EPS), oppure non viene raggiunto il numero massimo di iterazioni impostato.

Sistema Di Equzioni Lineari Ax=b

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Esempio Utilizzo metodo Jacobiano

1)Prendiamo questo sistema di equazioni lineari a tre incognite.

$$\begin{aligned} 4x_1 &- x_2 - x_3 = 3 \\ -2x_1 &+ 6x_2 + x_3 = 9 \\ -x_1 &+ x_2 + 7x_3 = -6 \end{aligned}$$

2)Successivamente sostituiamo un valore alle incognite, che sulla diagonale sia aumentato di 1 (ES. zero).

$$\begin{aligned} 4x_1^{(k+1)} &- x_2^{(k)} - x_3^{(k)} = 3 \\ -2x_1^{(k)} &+ 6x_2^{(k+1)} + x_3^{(k)} = 9 \\ -x_1^{(k)} &+ x_2^{(k)} + 7x_3^{(k+1)} = -6 \end{aligned}$$

3)Quindi otterremo questo sistema dove i valore non sulla diagonale saranno nulli.

$$\begin{aligned} 4x_1^{(1)} &- 0 - 0 = 3 \\ -2\cdot 0 &+ 6x_2^{(1)} + 0 = 9 \\ -0 &+ 0 + 7x_3^{(1)} = -6 \end{aligned} .$$

4)Dopo Questo passaggio otterremo che:

-X1=3/4

-x2=9/6

-x3=-6/7

Dopo aver iterato per 8 volte il metodo restituirà il valore esatto con l'approssimazione richiesta, dalla settima iterazione in poi il metodo restituirà sempre lo stesso valore.

k	$x^{(k)}$		
0	0.000	0.000	0.000
1	0.750	1.500	-0.857
2	0.911	1.893	-0.964
3	0.982	1.964	-0.997
4	0.992	1.994	-0.997
5	0.999	1.997	-1.000
6	0.999	2.000	-1.000
7	1.000	2.000	-1.000
8	1.000	2.000	-1.000

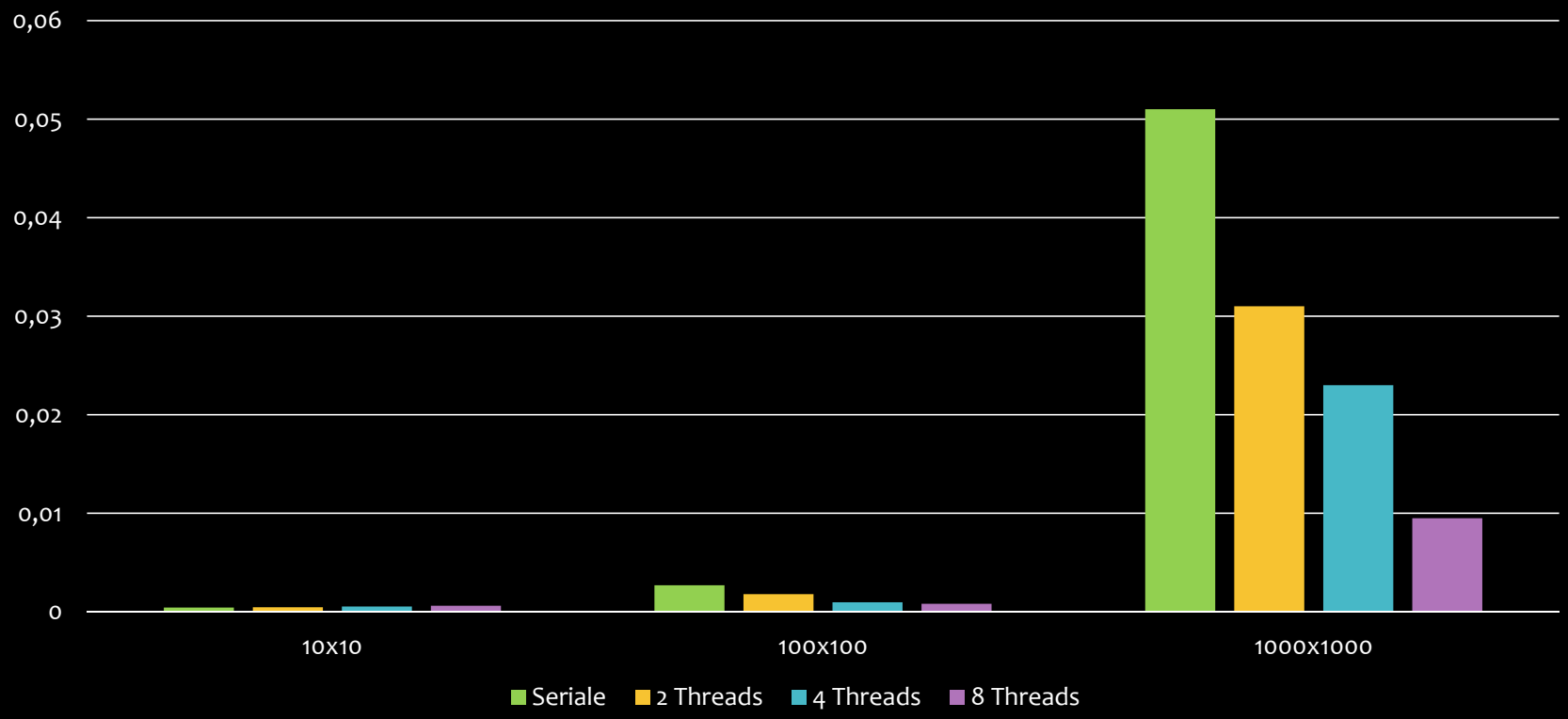
Complessità del Metodo Jacobiano

Il Metodo Jacobiano per la risoluzione di sistemi di equazioni lineari ha una complessità Equivalente a $O(N^2 \ln)$, è uguale a quella del metodo Gauss-Seidel, mentre rispetto all'eliminazione Gaussiana(Gauss-Jordan) è molto meno complesso in quanto ha una complessità $O(N^2)$.

Tempi Metodo di Jacobi

	Seriale	2 Threads	4 Threads	8 Threads
10x10	0,00043	0,00046	0,00054	0,00062
100x100	0,0027	0,0018	0,00097	0,00081
1000x1000	0,051	0,031	0,023	0,0095

Tempi di esecuzione metodo Jacobiano



Speed-up

- Nel Primo caso con una matrice molto piccola, possiamo notare come a parallelizzazione non porti benefici, anzi peggiori la velocità di esecuzione.
- Nel Secondo caso si può notare come il programma inizi a migliorare in quanto il metodo Jacobiano è di natura iterativa, quindi i casi da provare crescono molto e una parallelizzazione riesce a velocizzare i tempi di esecuzione.
- Il Terzo caso ci dimostra come all'aumentare della matrice dei coefficienti aumenta anche lo speed-up, che si avvicina alla linearizzazione.

	2 Threads	4 Threads	8 Threads
10x10	0,934	0,796	0,6935
100x100	1,5	2,78	3,33
1000x1000	1,64	2,25	5,36

```
65      first = my_rank*n / n_threads;
66      last = (my_rank + 1)*n / n_threads - 1;
67
68      for (i = 0; i < n_threads; ++i) { ... }
72
73      for (i = 0; i < n_threads; ++i) { ... }
77      start_time = MPI_Wtime();
78      MPI_Bcast(a, n*n, MPI_DOUBLE_PRECISION, 0, comm);
79      MPI_Bcast(b, n, MPI_DOUBLE_PRECISION, 0, comm);
80
81      max_x_err = 0.0;
82      for (i = 0; i < n; ++i) {
83          x[i] = b[i] / a[i*n + i];
84
85          if (abs(x[i]) > max_x_err) max_x_err = abs(x[i]);
86      }
87
88      k = 1;
89      max_row_err = 0.0;
90
91      do {
92          x_err = 0.0;
93
94
95          for (i = first; i <= last; ++i) {
96              x_new[i] = b[i];
97              for (j = 0; j < n; ++j) if (i != j) x_new[i] -= a[i*n + j] * x[j];
98              x_new[i] /= a[i*n + i];
99
100             if (x_err < abs(x_new[i] - x[i])) x_err = abs(x_new[i] - x[i]);
101         }
102
103         MPI_Allreduce(&x_err, &max_x_err, 1, MPI_DOUBLE_PRECISION, MPI_MAX, comm);
104
105         MPI_Alltoallv(x_new, x_count2, x_number2, MPI_DOUBLE_PRECISION, x, x_count, x_number, MPI_DOUBLE_PRECISION, comm);
106
107         row_err = 0.0;
108         sum = 0;
109
110         for (i = first; i <= last; ++i) {
111             sum = b[i];
112             for (j = 0; j < n; ++j) sum -= a[i*n + j] * x[j];
113
114             if (row_err < abs(sum)) row_err = abs(sum);
115         }
116
117         MPI_Allreduce(&row_err, &max_row_err, 1, MPI_DOUBLE_PRECISION, MPI_MAX, comm);
118
119         ++k;
120     } while ((max_x_err > EPS || max_row_err > EPS) && k <= max_iter);
```

Parallelizzazione Metodo di Jacobi

Nella parallelizzazione del metodo di Jacobi sono stati usati i costrutti più importanti di MPI per il passaggio di messaggio tra i threads. Inizialmente la matrice viene divisa tra tutti i threads presenti, successivamente viene effettuato un broadcast dove viene passato il sistema di equazioni lineari ai threads. Successivamente ogni threads effettua il metodo Jacobiano e si controlla se l'approssimazione massima di ogni iterazione è maggiore a quella richiesta. L'approssimazione massima viene ricavata tramite un AllReduce, in modo che ogni threads la abbia(dal momento che è una comunicazione molti a molti).