Slides Laboratorio Calcolo parallelo

Christian Barbato 771459
crbarbat@dsi.unive.it

17 settembre 2003

Problema: Sviluppare algoritmo parallelo per il calcolo distribuito dell'iterazione di Jacobi.

- Come suddividere la matrice tra i processi?
- In che modo rendere conoscenti i processi della topologia di suddivisione?
- Come far comunicare i processi per scambiarsi i dati?
- Affrontare politiche di bilanciamento del carico?
- Chi deve e come deve presentare i dati della computazione?

Idea: Sviluppare una serie di protocolli per risolvere le problematiche dell'applicazione distribuita.

La matrice è suddivisa in stripes (strisce) tutte di uguale altezza N (N dimensione del lato della matrice quadrata) e con larghezze x possibilmente uguali in fase di startup ma soprattutto tali che $\sum_i^n x_i = N$.

La suddivisione a strisce facilita l'implementazione dell'algoritmo in quanto risulta più facile stabilire i confini dei processi e l'eventuale migrazione delle stripes tra gli stessi al fine di bilanciare il carico.

Protocollo di riconoscimento dei confini.
E' un protocollo di tipo client-server.

1. Ogni processo spedisce a root l'indice delle stripes di confine da lui possedute.

2. Ogni processo spedisce a root l'indice delle strisce straniere di cui necessita (fase inutile nel caso di partizionamento contiguo).

3. Root spedisce a tutti i processi gli id dei processi in possesso delle stripes da loro richieste.

4. Root spedisce a tutti i processi il numero di clienti che dovranno servire durante il calcolo.

Protocollo per l'esecuzione di una iterazione di Jacobi.
Tutti i processi eseguono lo stesso codice.

1. Richiedo e attendo (procedure asincrone) le stripes agli altri processi.

2. Inizio atteso asincrona su richieste esterne.

3. Eseguo l'iterazione sulla parte di matrice che non necessita di dati esterni.

4. Servo le richieste altrui.

5. Attendo le stripes straniere per poter quindi completare l'iterazione.

6. Determino l'errore relativo locale e quello globale, conservo il valore più alto.

Protocollo per lo scambio dinamico delle strisce.
E' un protocollo client-server.

1. Ogni processo determina il tempo impiegato per l'esecuzione dell'ultima iterazione di Jacobi e lo spedisce a root.

2. Root calcola la media dei tempi di esecuzione e ne determina il più alto. Nel caso in cui il processo più lento superi la media, tenendo anche conto di una certa percentuale di tolleranza, ordina a tale processo di ridurre di una striscia il proprio carico di lavoro a favore del vicino. Più precisamente spedisce ad ogni processo un valore che può essere {-1, 1, 10, 0} a cui i processi reagiscono nella maniera seguente:

3. • -1: diminuire il carico di lavoro di una striscia (spedendola al vicino) e rieseguire il protocollo di riconoscimento dei confini.

   • 1: aumentare il carico di lavoro di una striscia e rieseguire il protocollo di riconoscimento dei confini.

   • 10: rieseguire il protocollo di riconoscimento dei confini.

   • 0: proseguire senza alcun cambiamento.

Protocollo per il recupero dei dati.

1. Ogni processo manda a root l'indice dei propri confini.

2. Ogni processo spedisce sotto forma di vettore i dati della propria computazione a root.

3. Root raccoglie le informazioni, crea la matrice totale, ne inserisce i dati e li presenta all'utente.

Sviluppo

Utilizzato linguaggio C++ per garantire una migliore suddivisione del codice (ereditarietà, isolamento degli errori) e per fornire una interfaccia singola per diverse implementazioni (polimorfismo).

Uso delle librerie MPICH. Da notare che ho cercato di utilizzare tutti i tipi di comunicazione:

- Send e Receive bloccanti nel protocollo di riconoscimento delle strisce.

- Send e Receive non bloccanti nel calcolo di una iterazione dell'algoritmo.

- Chiamate collettive (Scatter, Gather) nel protocollo di scambio dinamico.

Possibili miglioramenti

- Utilizzo dei comunicatori.
- Migliorare il protocollo di scambio dinamico (più di una striscia alla volta e tra processi non contigui).
- Partizionamento a blocchi.
- Renderlo fault tolerant.
- Utilizzare librerie che permettano VSM (Virtual Shared Memory) o che siano conformi a MPI-2 potendo quindi utilizzare la creazione dinamica dei processi.

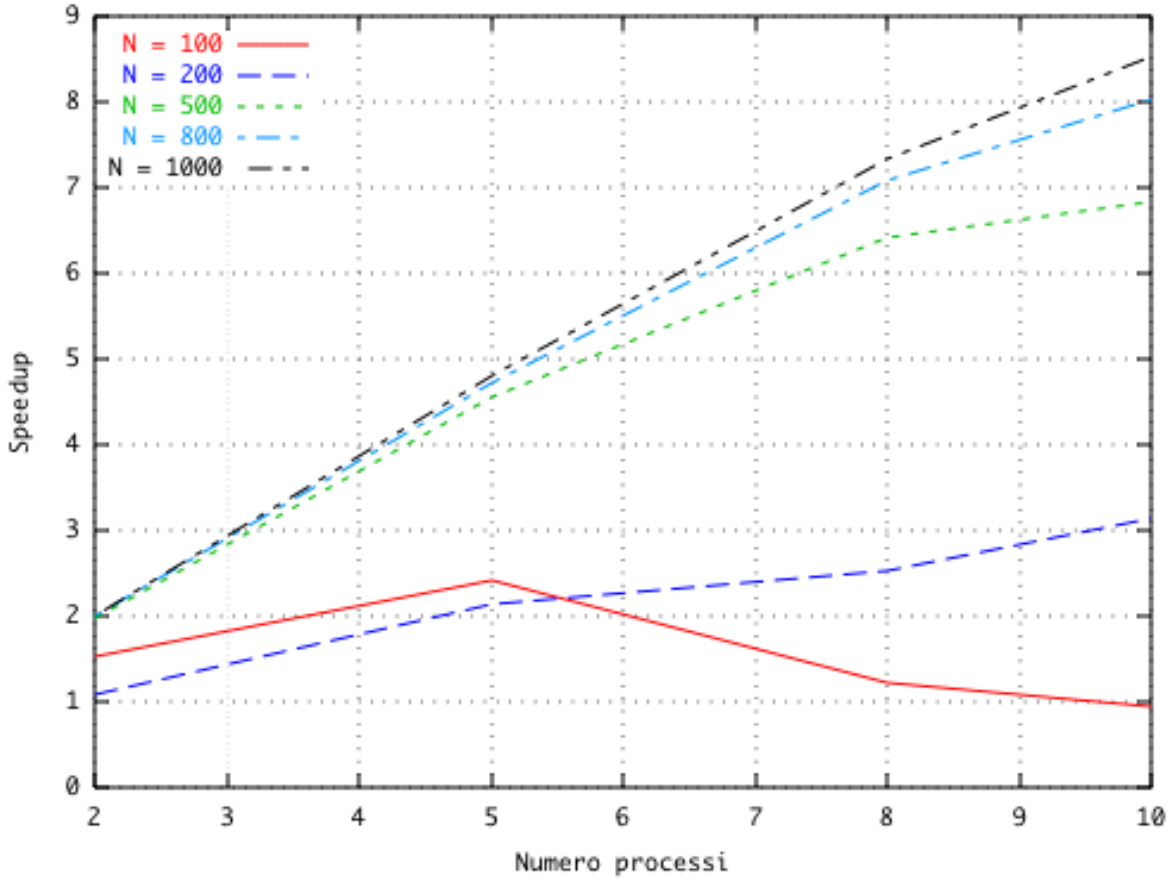
Risultati algoritmo sequenziale

Dimensione lato matrice	Tolleranza	Tempo (secondi)
100	0.1	2.5
200	0.1	11.03
500	0.1	65.32
800	0.1	165.19
1000	0.1	257.15
100	0.01	18.08
200	0.01	102
500	0.01	744.76
800	0.01	1793.45
1000	0.01	2585.30

Notare il decadimento delle prestazioni tra gli ultimi due punti.

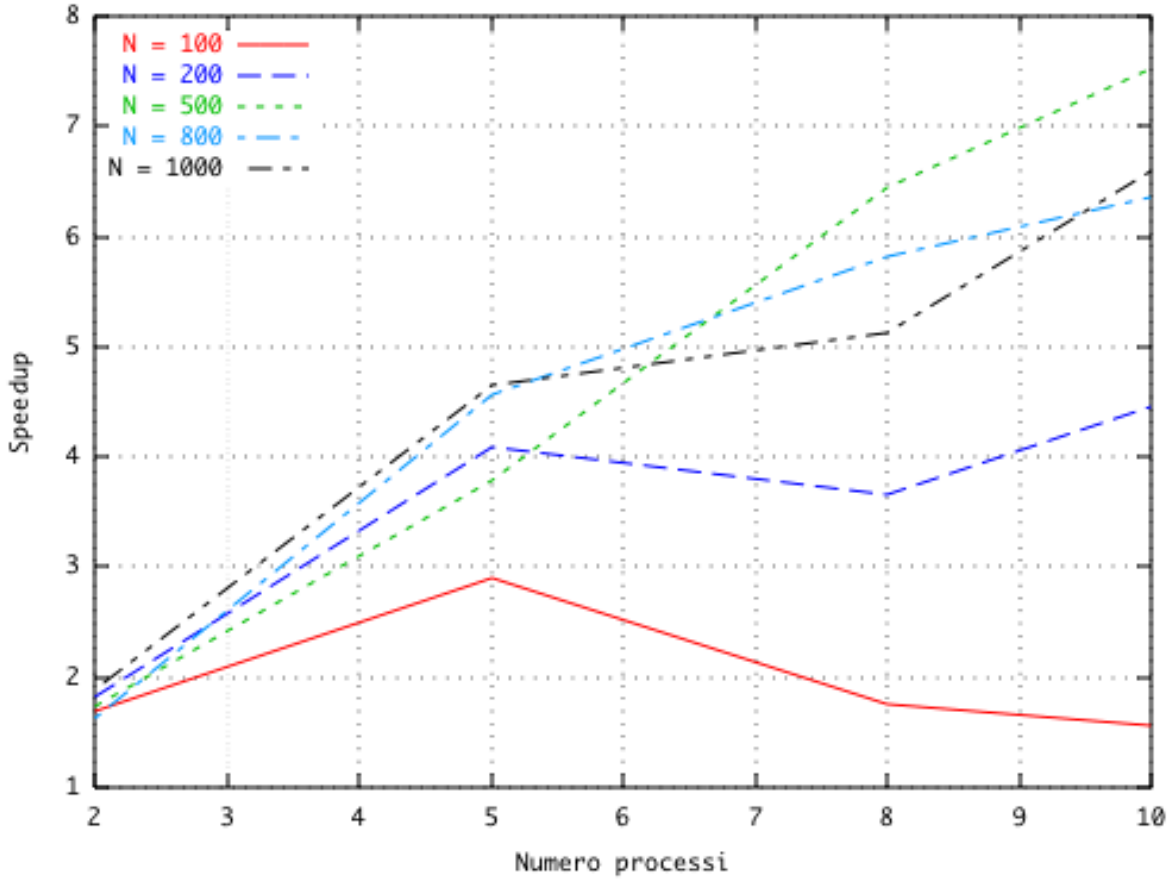
Speedup algoritmo parallelo statico (1/3)

Tolleranza 0.1, passo di sincronia unitario.



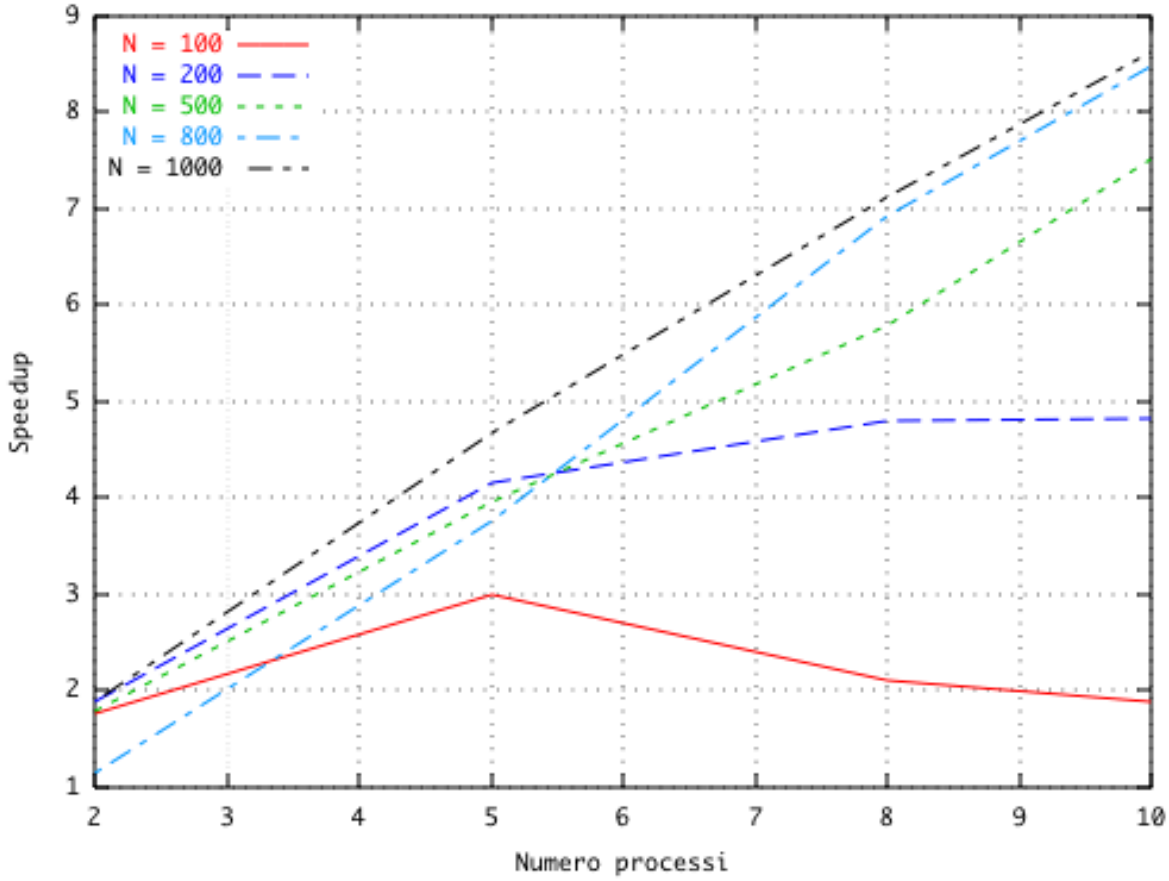
Speedup algoritmo parallelo statico (2/3)

Tolleranza 0.1, passo di sincronia 5.



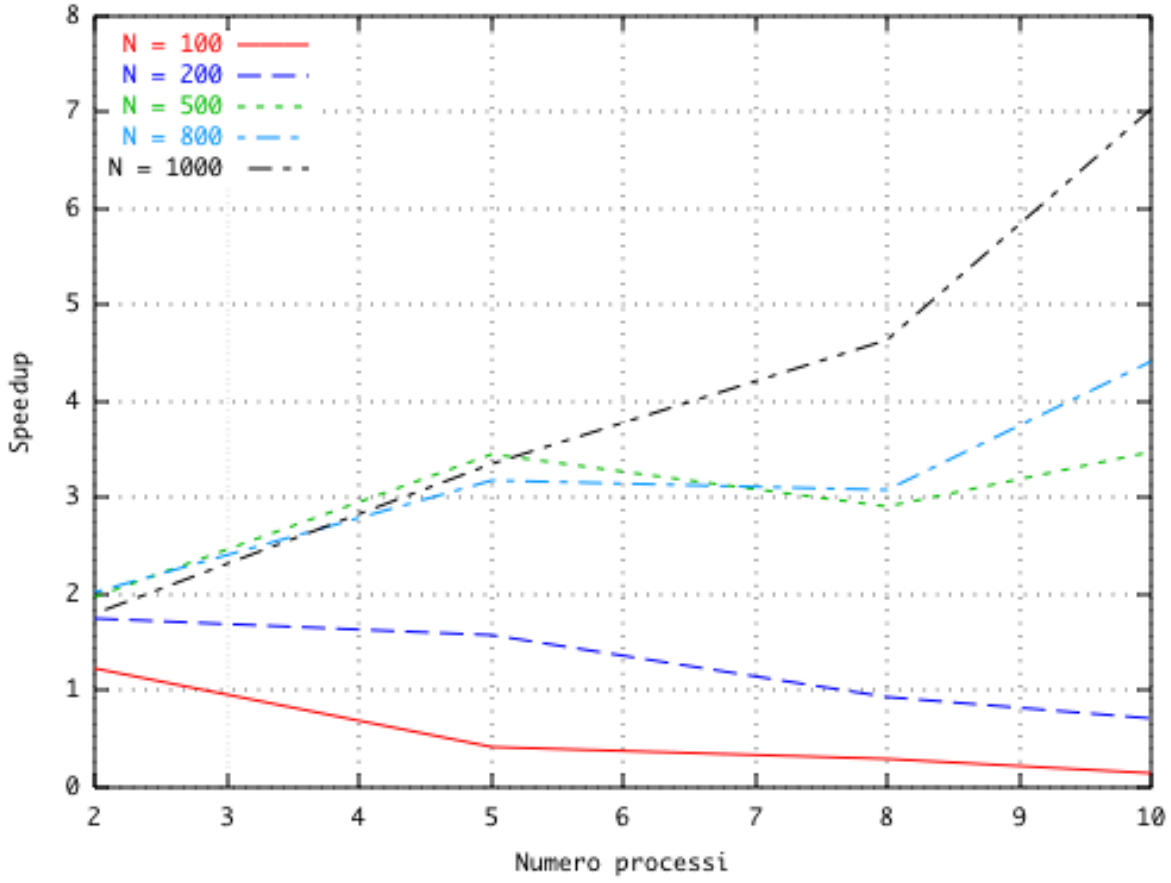
Speedup algoritmo parallelo statico (3/3)

Tolleranza 0.1, passo di sincronia 10.



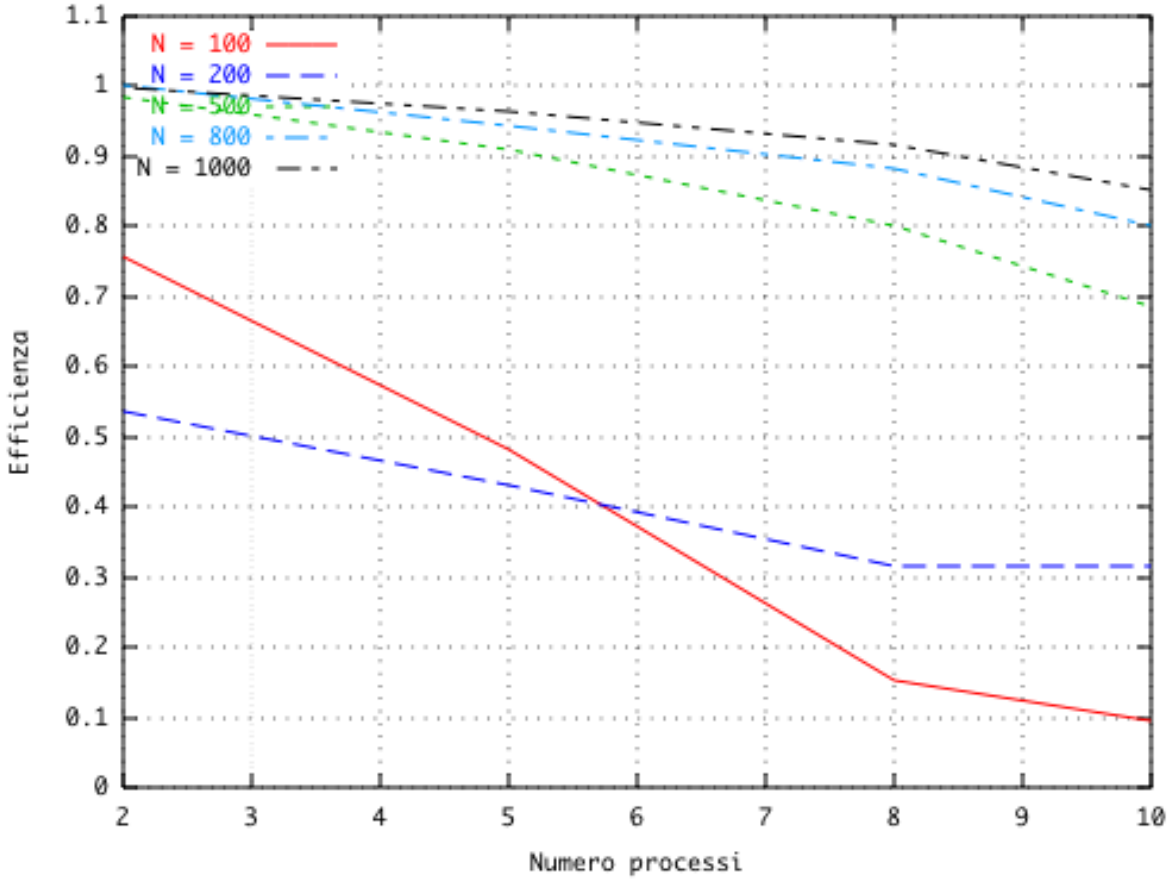
Speedup algoritmo parallelo dinamico

Tolleranza 0.1, passo di sincronia 1.



Efficienza algoritmo parallelo statico

Tolleranza 0.1, passo di sincronia 1.



Efficienza algoritmo parallelo dinamico

Tolleranza 0.1, passo di sincronia 1.