Disegnare programmi paralleli

Parallelizzazione automatica versus manuale

- Parallelizzazione manuale, complessa e costosa
- Parallelizzazione auotamica: completamente automatica (i loop sono i target piu' frequenti per la parallelizzazione automatica) o mediante l'uso di direttive al compilatore

Disegnare programmi paralleli

Capire il problema ed il programma

- Capire se il problema e' parallelizzabile o meno. Ad esempio la serie di Fibonacci:

  $$F(K+2) = F(K + 1) + F(K)$$

  In questo caso visto che F(K+2) dipende sia dal termine K + 1 che K, e' chiaro che questi tre termini non possono essere calcoati indipendetemente, quindi non in parallelo. Considerata invece i casi visti del calcolo di PI.
- Fare il profiling (ex. Gprof), determinate gli “hotspots”, i punti dove viene fatta la maggior parte del lavoro.
- Determinare i “bottleneck” parti che sono molto lente, tipicamente I/O.

Disegnare programmi paralleli

- Identificare inibitori del parallelismo (ad esempio dipendenza dei dati tipo serie di fibonacci)

- **Ma soprattutto cerchiamo di capire se e quanto e' possibile migliorare l'algoritmo sequenziale (vedi lezione 6) e/o se esistono algoritmi sequenziali migliori**

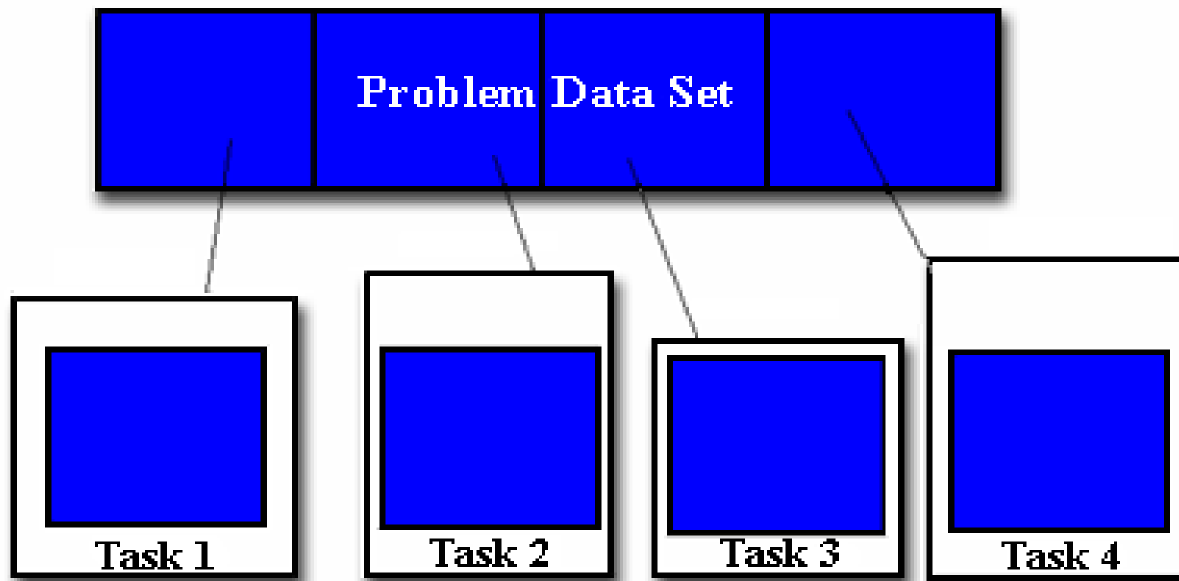
Disegnare programmi paralleli

Partitioning: Una dei primi steps e' dividere il problema in porzioni discrete di lavoro. Porzioni che possano essere agevolmente distribuite. Ci sono fondamentalmente due modi di decomporre un lavoro di tipo computazionale:

- **domain decomposition**
- **functional decomposition.**

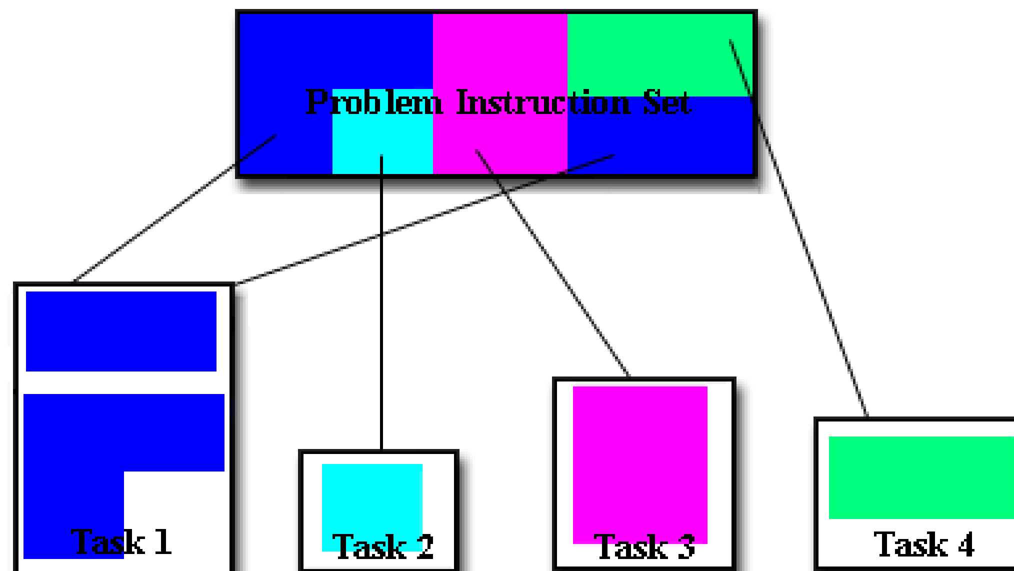
Domain Decomposition

- I dati associati al problema vengono decomposti, cosi' che ogni task parallelo puo' lavorare in una pozione di dati.
- Tutti eseguono le stesse operazioni ma su un sottoinsieme differente dei dati (SPMD)
- Pro [Scalabile con quantita' di dati] Contro [Generalemnte vantaggioso solo per grandi quantita' di dati]



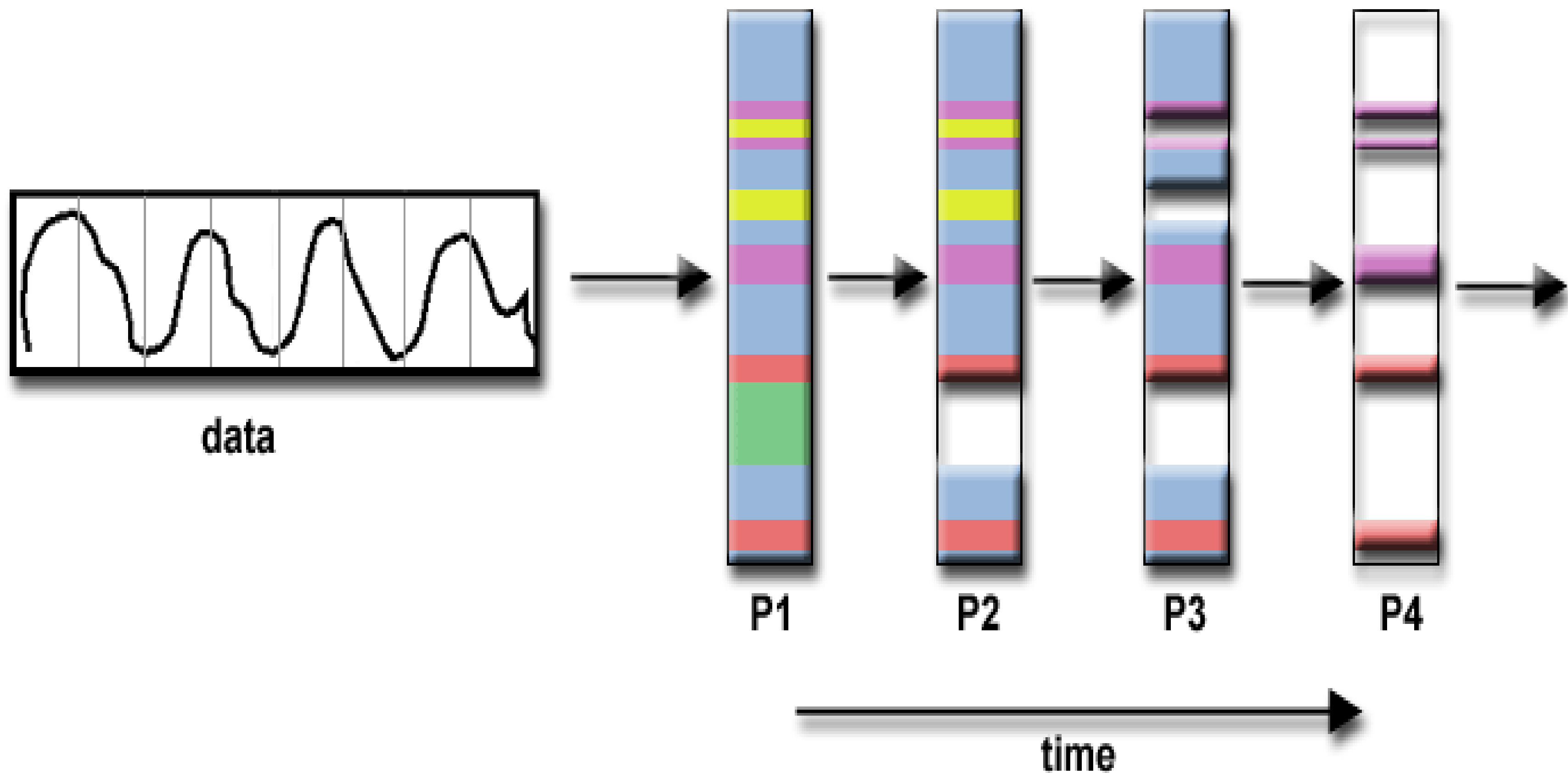
Functional Decomposition

- Distribuzione delle funzioni tra piu' soggetti. Decompongo il lavoro in base al lavoro che deve essere svolto
- Ogni task prende in carico un particolare elaborazione (MPMD)
- Pro [scalabile con il numero di elaborazioni indipendenti] Contro [vantaggioso solo per elaborazioni sufficientemente complesse]



Functional Decomposition

Immaginate di dover applicare ad un segnale differnti filtri uno di seguito all'altro (pipeline)



Comunicazione

E' necessaria ? Dipende dal problema:

- Non e' necessaria se il calcolo e' imbarazzantemente parallelo
- Si e' necessria, nella maggior parte dei casi ogni task deve condividere dati con gli altri.

Costi della comunicazione:

- Cicli macchina che potrebbero essere usati per calcolore, sono in realta' spesi a spedire e riceve dati
- La comunicazione richiede in genrale qualche tipo di sincronizzazione, e quindi qualche task spendera' tempo ad aspettare
- Comunicazioni sovrapposte possono prima o poi saturare la banda a disposizione
- Latenza (spesso, quando e' possibile, conviene raggruppare tanti piccoli dati e spedirli assieme)

Load balancing

E' importante per le performance dei programmi paralleli

- Distribuire il carico di lavoro in modo che tutti i task siano occupati per tutto il tempo
- Minimizzare i tempi morti (tempi di attesa) dei task
- Partizionare ugualmente il lavoro di ogni task
  - Assegnazione statica, esempi:
    - Per operazioni su array/matrici dove ogni task svolge operazioni simili, uniformemente distribuisco i dati sui task
    - Per cicli dove il lavoro fatto in ogni iterazione e' simile, uniformemente distribuisco le iterazioni sui task

Load balancing: problema tipico

Ho 4 processi:

.tempo seriale = 40
.tempo parallelo (teorico) = 10

.tempo parallelo (reale) = 12 - 20% più lento
.la velocita' dell'esecuzione dipendera' dal processo piu' lento

Load balancing: Assegnazione dinamica

Alcuni partizionamenti anche se uniformi risultano sempre non bilanciati:

- Matrici sparse
- Metodi con griglie adattative
- Simulazioni N-corpi

In questi casi solo l'assegnazione dinamica puo' riuscire a bilanciare il carico computazionale

- Divido la regione in tante parti, ogni task prendera' in carico una parte
- Quando un task termina una parte, ne richiede una nuova

Load balancing: confronti

Assegnazione statica:

- in genere semplice, proporzionale al volume
- soffre di possibili sbilanciamenti

Assegnazione dinamica:

- puo' curare problemi di sbilanciamenti
- introduce un overhead dovuto alla gestione del bilanciamento

Granularita'

Misura qualitativa del rapporto tra calcoli e comunicazioni

Parallelismo a grana fine

- Pochi conti tra le comunicazioni rapporto piccolo
- Puo' essere facile bilanciare il carico
- Overhead di comunicazioni

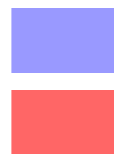
Parallelismo a grana grossa

- Molti conti tra le comunicazioni rapporto grande
- Puo' essere difficile bilanciare il carico
- Probabili aumenti nelle performance

Granularita'

time

time



communication

computation

Input e Output

- Le operazioni di I/O sono generalmente seriali e possono creare “colli di bottiglia”
- Operazioni di I/O sono in generale trattate some inibitori del parallelismo
- In un ambiante di lavoro in cui i vari task vedono lo stesso filespace, le operazioni di writing porteranno ad una sovrascrittura del file
- Le operazioni di lettura saranno comunque limitate dall'abilita' del fileserver di gestire piu' operazioni di read contemporaneamente
- I/O over NFS rappresenta comunque un collo di bottiglia.

Input e Output

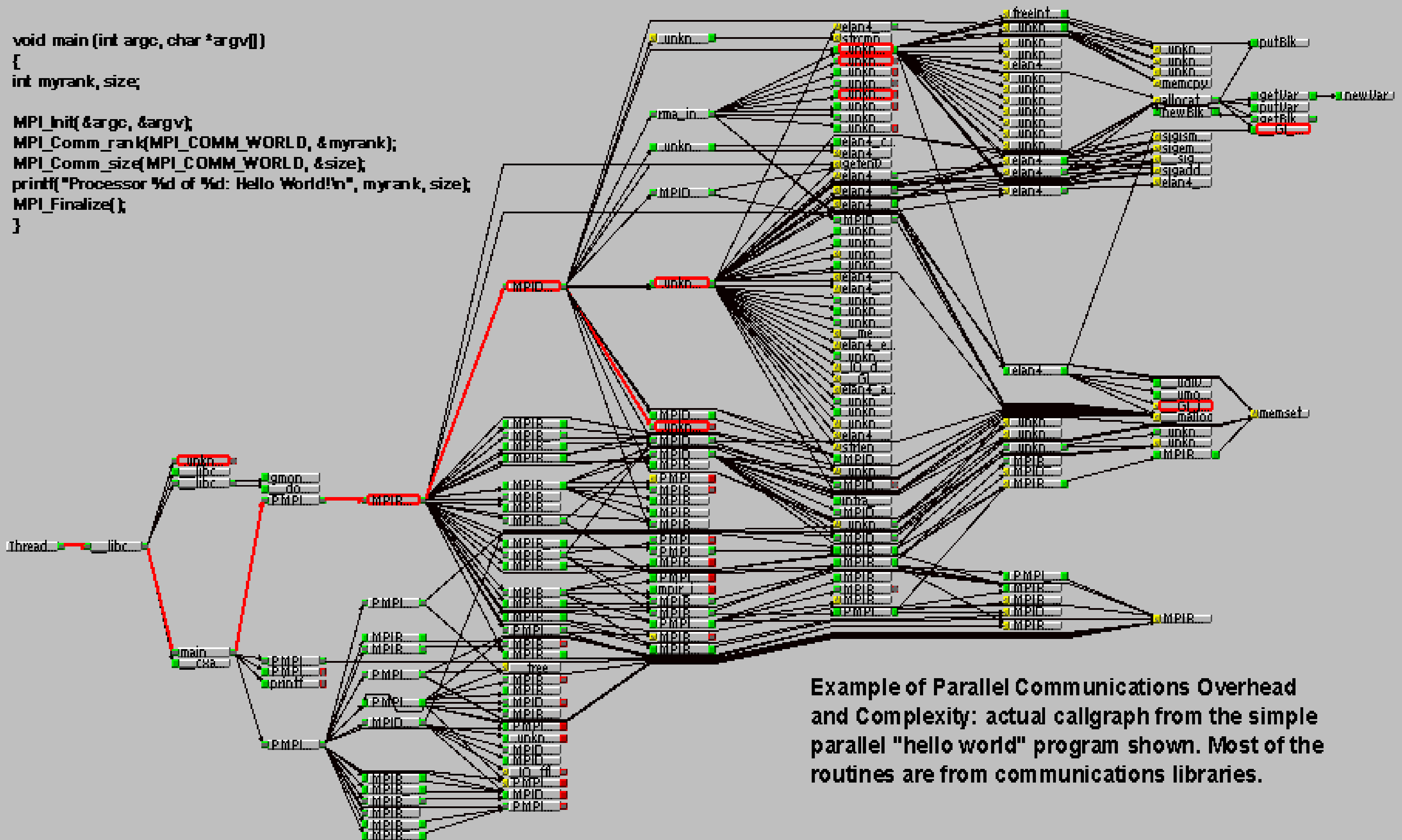
Tuttavia ci sono alcune vie di uscita:

- Filesystem paralleli (GPFS, Luster, PVFS/PVFS2, PanFS, HP SFS)
- MPI-2 prevede un'interfaccia per l'I/O parallelo

Altrimenti:

- Ridurre l'i/O al minimo indispensabile
- Confinare l'I/O su porzioni seriali del codice, poi Gather/Scatter per rimettere insieme/distribuire i dati su modelli message-passing. Utilizzo di un solo thread nel modello di programmazione multi-threaded
- Per sistemi a memoria distribuita tipo i COW e' meglio fare I/O nei filespace locali piuttosto che ne filespace condivisi (chemgrid: /u0 /u1 piuttosto che /home)
- Creare filnemane unici per ogni tasks

Overhead and Complexity



Example of Parallel Communications Overhead and Complexity: actual callgraph from the simple parallel "hello world" program shown. Most of the routines are from communications libraries.