



Design of Parallel Algorithm

Programmazione Concorrente, Parallela e su Cloud

Carmine Spagnuolo, Ph.D.



Plan



1 Progettazione di Programmi

- Tecniche
- Le problematiche
- Esempio 1: Array processing
- Esempio 2: Heat equation



Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*

Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica

Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore


Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore
- Compilatore *fully automatic*


Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore
- Compilatore *fully automatic*
 - analizza il codice ed identifica opportunità con calcolo dei pesi (esempio: loops)


Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
 - Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore
 - Compilatore *fully automatic*
 - analizza il codice ed identifica opportunità con calcolo dei pesi (esempio: loops)
-  possibili errori, mancanza flessibilità, limitato a parti del codice



Parallellizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore
- Compilatore *fully automatic*
 - analizza il codice ed identifica opportunità con calcolo dei pesi (esempio: loops)
 -  possibili errori, mancanza flessibilità, limitato a parti del codice
- Compilatore *programmer directed*

Parallelizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore
- Compilatore *fully automatic*
 - analizza il codice ed identifica opportunità con calcolo dei pesi (esempio: loops)
 -  possibili errori, mancanza flessibilità, limitato a parti del codice
- Compilatore *programmer directed*
 - direttive al compilatore per aiutarlo a parallelizzare

Parallellizzazione di programmi

- Tipicamente una operazione manuale: compito complesso, *error-prone*
- Esistono tool per la parallelizzazione automatica
 - che assistono il programmatore
- Compilatore *fully automatic*
 - analizza il codice ed identifica opportunità con calcolo dei pesi (esempio: loops)
 -  possibili errori, mancanza flessibilità, limitato a parti del codice
- Compilatore *programmer directed*
 - direttive al compilatore per aiutarlo a parallelizzare
 -  difficile da usare

Uno sguardo al problema/programma

- Problema parallelizzabile o no

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale
- Identificare gli *hotspots* del programma

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale
- Identificare gli *hotspots* del programma
 - Uso di profiler per identificare dove viene fatto la maggior parte del lavoro

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale
- Identificare gli *hotspots* del programma
 - Uso di profiler per identificare dove viene fatto la maggior parte del lavoro
- Identifica i *bottlenecks*

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale
- Identificare gli *hotspots* del programma
 - Uso di profiler per identificare dove viene fatto la maggior parte del lavoro
- Identifica i *bottlenecks*
 - Sincronizzazione, I/O

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale
- Identificare gli *hotspots* del programma
 - Uso di profiler per identificare dove viene fatto la maggior parte del lavoro
- Identifica i *bottlenecks*
 - Sincronizzazione, I/O
- Identificare gli inibitori del parallelismo

Uno sguardo al problema/programma

- Problema parallelizzabile o no
 - Esempio per Fibonacci: il calcolo di F_{k+2} usa valori di F_{k+1} e F_k . Soluzione non banale
- Identificare gli *hotspots* del programma
 - Uso di profiler per identificare dove viene fatto la maggior parte del lavoro
- Identifica i *bottlenecks*
 - Sincronizzazione, I/O
- Identificare gli inibitori del parallelismo
 - dipendenze tra dati

Presentazione



- 1 Progettazione di Programmi
 - Tecniche
 - Le problematiche
 - Esempio 1: Array processing
 - Esempio 2: Heat equation



Partitioning



- Uno dei primi passi é dividere il problema in “pezzi” discreti di lavoro



Partitioning

- Uno dei primi passi é dividere il problema in “pezzi” discreti di lavoro
- ...che possono essere distribuiti

Partitioning

- Uno dei primi passi é dividere il problema in “pezzi” discreti di lavoro
- ... che possono essere distribuiti
- Due maniere fondamentale di fare partitioning:

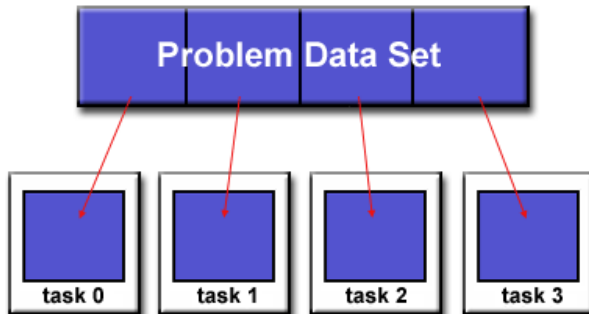
Partitioning

- Uno dei primi passi é dividere il problema in “pezzi” discreti di lavoro
- ... che possono essere distribuiti
- Due maniere fondamentale di fare partitioning:
 - Domain decomposition: i dati che appartengono al problema vengono decomposti ed ogni task lavora su una porzione dei dati

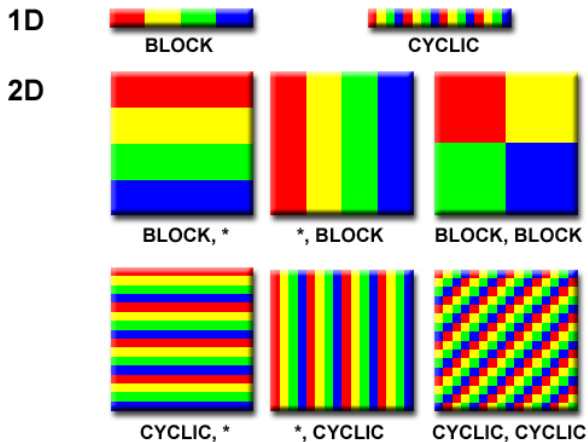
Partitioning

- Uno dei primi passi é dividere il problema in “pezzi” discreti di lavoro
- ... che possono essere distribuiti
- Due maniere fondamentale di fare partitioning:
 - Domain decomposition: i dati che appartengono al problema vengono decomposti ed ogni task lavora su una porzione dei dati
 - Functional decomposition: decomposizione secondo il tipo di lavoro da fare, assegnato a ciascun task

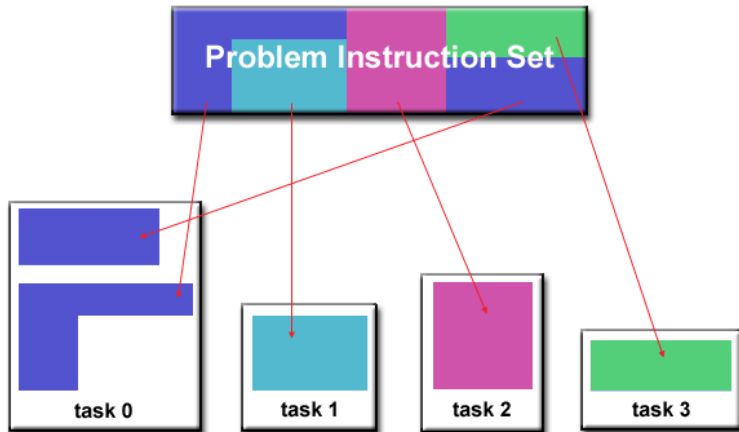
Domain decomposition - 1



Domain decomposition - 2

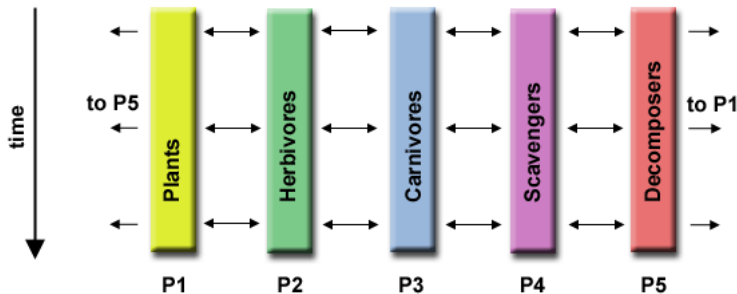


Functional decomposition



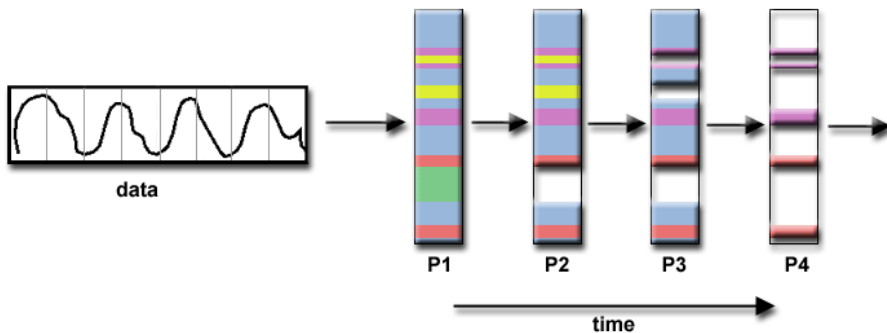
Functional decomposition: esempio - 1

Modellazione di un ecosistema



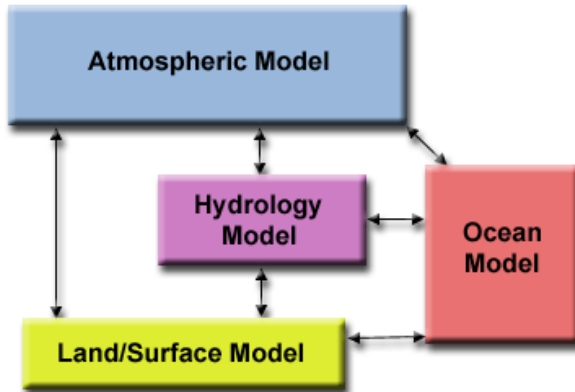
Functional decomposition: esempio - 2

Signal processing in pipeline



Functional decomposition: esempio - 3

Modello del clima



Presentazione



- 1 Progettazione di Programmi
 - Tecniche
 - Le problematiche
 - Esempio 1: Array processing
 - Esempio 2: Heat equation



La comunicazione

- Dato il problema, potrebbe non essere necessaria

La comunicazione

- Dato il problema, potrebbe non essere necessaria
 - Tipico partizionamento di una immagine in regioni, con calcoli in locale

La comunicazione

- Dato il problema, potrebbe non essere necessaria
 - Tipico partizionamento di una immagine in regioni, con calcoli in locale
- ... oppure potrebbe essere necessaria

La comunicazione

- Dato il problema, potrebbe non essere necessaria
 - Tipico partizionamento di una immagine in regioni, con calcoli in locale
- ... oppure potrebbe essere necessaria
- Fattori da considerare

La comunicazione

- Dato il problema, potrebbe non essere necessaria
 - Tipico partizionamento di una immagine in regioni, con calcoli in locale
- ... oppure potrebbe essere necessaria
- Fattori da considerare
 - influenzano le scelte del programmatore nella progettazione

Comunicazione: i fattori - 1

- Costo di comunicazione

Comunicazione: i fattori - 1

- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare

Comunicazione: i fattori - 1

- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare
 - necessario a volte sincronizzare (bottleneck)

Comunicazione: i fattori - 1

- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare
 - necessario a volte sincronizzare (bottleneck)
 - competizione per risorse limitate (bus)

Comunicazione: i fattori - 1

- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare
 - necessario a volte sincronizzare (bottleneck)
 - competizione per risorse limitate (bus)
- Latency vs. Bandwidth

Comunicazione: i fattori - 1

- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare
 - necessario a volte sincronizzare (bottleneck)
 - competizione per risorse limitate (bus)
- Latency vs. Bandwidth
 - facendo il package di molti messaggi piccoli in un unico messaggio grande, può risultare più efficiente

Comunicazione: i fattori - 1

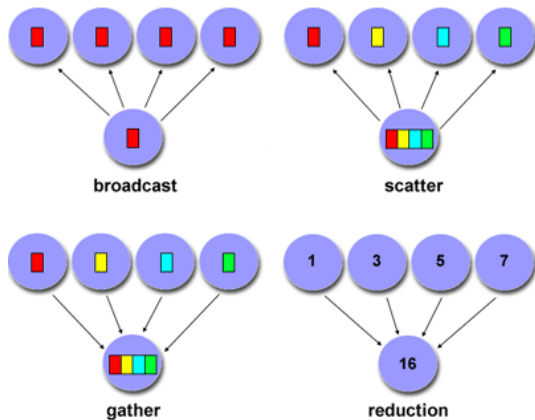
- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare
 - necessario a volte sincronizzare (bottleneck)
 - competizione per risorse limitate (bus)
- Latency vs. Bandwidth
 - facendo il package di molti messaggi piccoli in un unico messaggio grande, può risultare più efficiente
- Visibilità della comunicazione (esplicita o implicita)

Comunicazione: i fattori - 1

- Costo di comunicazione
 - cicli macchina e risorse sono usate per comunicare invece che per calcolare
 - necessario a volte sincronizzare (bottleneck)
 - competizione per risorse limitate (bus)
- Latency vs. Bandwidth
 - facendo il package di molti messaggi piccoli in un unico messaggio grande, può risultare più efficiente
- Visibilità della comunicazione (esplicita o implicita)
- Sincrona/Asincrona (blocking/non-blocking)

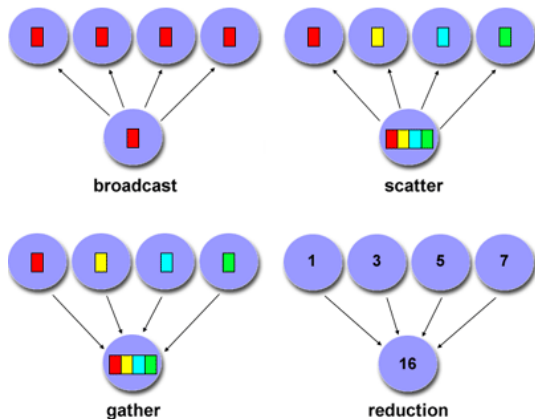
Comunicazione: i fattori - 2

- *Scope della comunicazione:*
point-to-point o collective



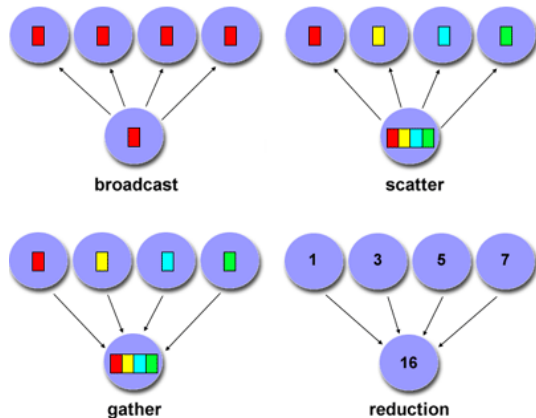
Comunicazione: i fattori - 2

- *Scope* della comunicazione:
point-to-point o collective
- Operazioni diverse:



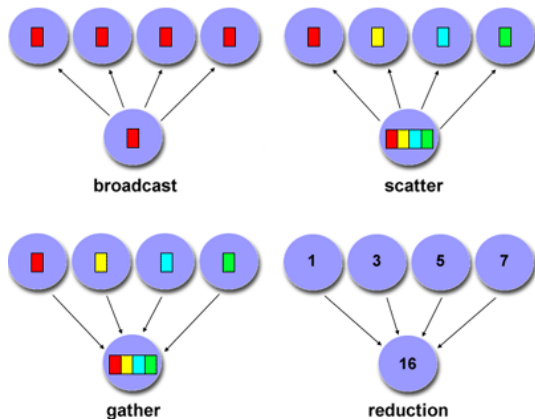
Comunicazione: i fattori - 2

- *Scope* della comunicazione: point-to-point o collective
- Operazioni diverse:
- **broadcast**



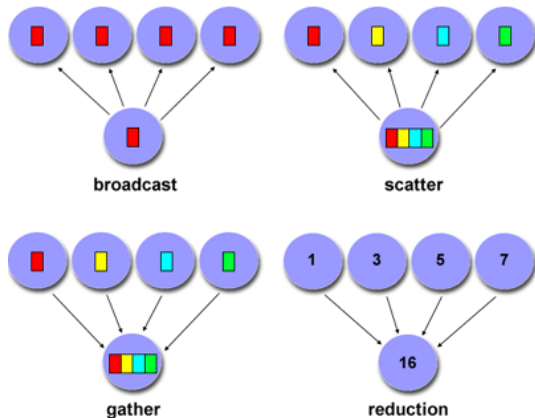
Comunicazione: i fattori - 2

- *Scope* della comunicazione: point-to-point o collective
- Operazioni diverse:
- broadcast
- scatter
- gather
- reduction



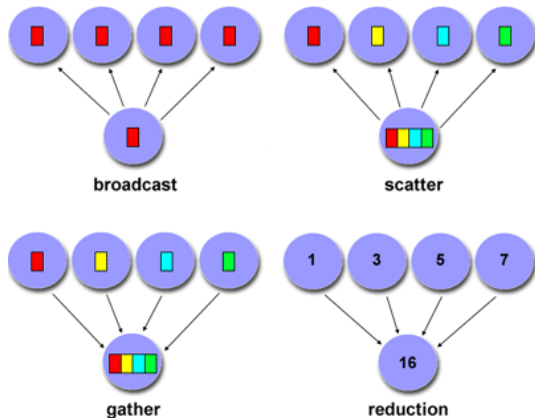
Comunicazione: i fattori - 2

- *Scope* della comunicazione: point-to-point o collective
- Operazioni diverse:
- broadcast
- scatter
- gather
- reduction



Comunicazione: i fattori - 2

- *Scope* della comunicazione:
point-to-point o collective
- Operazioni diverse:
- broadcast
- scatter
- gather
- reduction



Comunicazione: i fattori - 3: efficienza

- Dipendenza da implementazioni efficienti su una particolare piattaforma

Comunicazione: i fattori - 3: efficienza

- Dipendenza da implementazioni efficienti su una particolare piattaforma
- Efficienza della comunicazione asincrona rispetto a quella sincrona

Sincronizzazione



- Barrier: ogni task si blocca alla barriera, fino a che tutti sono lì, in attesa



Sincronizzazione

- Barrier: ogni task si blocca alla barriera, fino a che tutti sono lì, in attesa
- Lock/semaforo: protegge sezioni critiche

Sincronizzazione

- Barrier: ogni task si blocca alla barriera, fino a che tutti sono lì, in attesa
- Lock/semaforo: protegge sezioni critiche
- Operazioni sincrone di comunicazione

Dipendenza dei dati

- Esiste dipendenza tra istruzioni se l'ordine di esecuzione influenza il risultato

Dipendenza dei dati

- Esiste dipendenza tra istruzioni se l'ordine di esecuzione influenza il risultato
- Esiste dipendenza dati se task differenti usano contemporaneamente gli stessi dati

Dipendenza dei dati

- Esiste dipendenza tra istruzioni se l'ordine di esecuzione influenza il risultato
- Esiste dipendenza dati se task differenti usano contemporaneamente gli stessi dati
- Gestione:

Dipendenza dei dati

- Esiste dipendenza tra istruzioni se l'ordine di esecuzione influenza il risultato
- Esiste dipendenza dati se task differenti usano contemporaneamente gli stessi dati
- Gestione:
 - su memoria distribuita: comunicazione sulle richieste di dati a punti di sincronizzazione

Dipendenza dei dati

- Esiste dipendenza tra istruzioni se l'ordine di esecuzione influenza il risultato
- Esiste dipendenza dati se task differenti usano contemporaneamente gli stessi dati
- Gestione:
 - su memoria distribuita: comunicazione sulle richieste di dati a punti di sincronizzazione
 - su memoria condivisa: read/write sincronizzate

Esempi di dipendenza

- Il valore di $A[j - 1]$ deve essere calcolato prima di $A[j]$: *data dependence* dovuta a loop

```
DO 500 J = MYSTART,MYEND  
  A(J) = A(J-1) * 2.0  
500 CONTINUE
```

task 1 -----	task 2 -----
X = 2	X = 4
.	.
.	.
Y = X**2	Y = X**3

Esempi di dipendenza

- Il valore di $A[j - 1]$ deve essere calcolato prima di $A[j]$: *data dependence* dovuta a loop
 - se i due valori sono su task diversi, necessaria sincronizzazione e comunicazione

```
DO 500 J = MYSTART,MYEND  
  A(J) = A(J-1) * 2.0  
500 CONTINUE
```

task 1	task 2
-----	-----
X = 2	X = 4
.	.
.	.
Y = X**2	Y = X**3

Esempi di dipendenza

- Il valore di $A[j - 1]$ deve essere calcolato prima di $A[j]$: *data dependence* dovuta a loop
 - se i due valori sono su task diversi, necessaria sincronizzazione e comunicazione
- Dipendenza non dovuta a loop

```
DO 500 J = MYSTART,MYEND  
  A(J) = A(J-1) * 2.0  
500 CONTINUE
```

task 1	task 2
-----	-----
X = 2	X = 4
.	.
.	.
Y = X**2	Y = X**3

Esempi di dipendenza

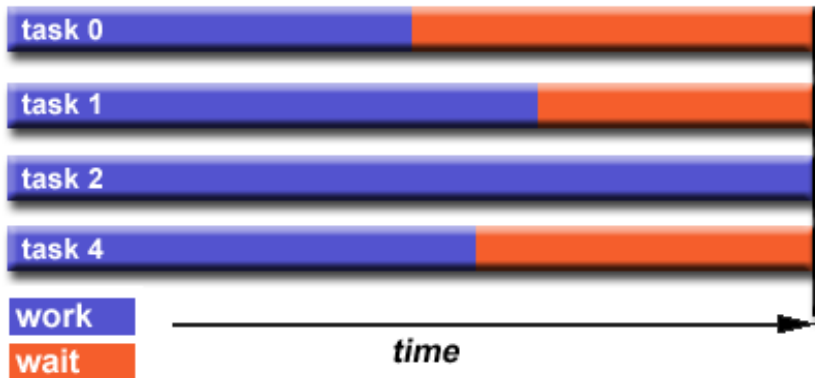
- Il valore di $A[j - 1]$ deve essere calcolato prima di $A[j]$: *data dependence* dovuta a loop
 - se i due valori sono su task diversi, necessaria sincronizzazione e comunicazione
- Dipendenza non dovuta a loop
 - il valore di Y dipende da comunicazione tra task o sequenzializzazione scritture su X

```
DO 500 J = MYSTART,MYEND  
  A(J) = A(J-1) * 2.0  
500 CONTINUE
```

task 1	task 2
-----	-----
X = 2	X = 4
.	.
.	.
Y = X**2	Y = X**3

Bilanciamento del carico

Importante che tutti i task siano occupati per tutto il tempo



Come bilanciare il carico

- Equipartizionare il lavoro: a volte facile, a volte impossibile

Come bilanciare il carico

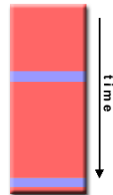
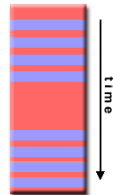
- Equipartizionare il lavoro: a volte facile, a volte impossibile
 - macchine eterogenee, work eterogeneo ed imprevedibile

Come bilanciare il carico

- Equipartizionare il lavoro: a volte facile, a volte impossibile
 - macchine eterogenee, work eterogeneo ed imprevedibile
- Assegnamento dinamico del lavoro: con uno scheduler, a cui i task richiedono un altro batch di lavoro

Granularità

- Granularità: rapporto tra computazione e comunicazione

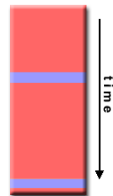
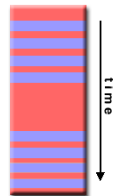


communication
computation



Granularità

- Granularità: rapporto tra computazione e comunicazione
- Parallelismo *fine-grain*

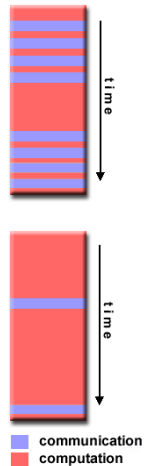


communication
computation



Granularità

- Granularità: rapporto tra computazione e comunicazione
- Parallelismo *fine-grain*
- Parallelismo *coarse-grain*



Fine vs. Coarse

- Fine-grain:



Fine vs. Coarse

- Fine-grain:

 facilita il bilanciamento

Fine vs. Coarse

- Fine-grain:



facilita il bilanciamento



alto overhead di comunicazione

Fine vs. Coarse

- Fine-grain:



facilita il bilanciamento





alto overhead di comunicazione

- Coarse-grain:

Fine vs. Coarse

- Fine-grain:

-  facilita il bilanciamento
-  alto overhead di comunicazione

- Coarse-grain:

-  maggiori opportunità di miglioramento prestazioni (minor overhead di comunicazione)

Fine vs. Coarse

- Fine-grain:



facilita il bilanciamento



alto overhead di comunicazione

- Coarse-grain:



maggiori opportunità di miglioramento prestazioni (minor overhead di comunicazione)



carico difficile da bilanciare

Input/Output

- In generale, I/O é un problema per il parallelismo

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)
- Esistono sistemi paralleli di file system

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)
- Esistono sistemi paralleli di file system
- Possibili ottimizzazioni particolari (Google File system)

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)
- Esistono sistemi paralleli di file system
- Possibili ottimizzazioni particolari (Google File system)
- Linee guida:

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)
- Esistono sistemi paralleli di file system
- Possibili ottimizzazioni particolari (Google File system)
- Linee guida:
 - ① se possibile, evitatelo

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)
- Esistono sistemi paralleli di file system
- Possibili ottimizzazioni particolari (Google File system)
- Linee guida:
 - 1 se possibile, evitatelo
 - 2 sostituire accesso parallelo ad un accesso sequenziale seguito dalla distribuzione dei dati

Input/Output

- In generale, I/O é un problema per il parallelismo
- Specialmente se condotto sulla rete (NFS)
- Esistono sistemi paralleli di file system
- Possibili ottimizzazioni particolari (Google File system)
- Linee guida:
 - 1 se possibile, evitatelo
 - 2 sostituire accesso parallelo ad un accesso sequenziale seguito dalla distribuzione dei dati
 - 3 rendere i file univoci

Il costo del calcolo parallelo

- Complessità:

Il costo del calcolo parallelo

- Complessità:
 - progettazione, codifica, debugging, tuning, manutenzione

Il costo del calcolo parallelo

- Complessità:
 - progettazione, codifica, debugging, tuning, manutenzione
- Portabilità

Il costo del calcolo parallelo

- Complessità:
 - progettazione, codifica, debugging, tuning, manutenzione
- Portabilità
 - migliorata con standard, ma ancora un problema (OSs, HW)

Il costo del calcolo parallelo

- Complessità:
 - progettazione, codifica, debugging, tuning, manutenzione
- Portabilità
 - migliorata con standard, ma ancora un problema (OSs, HW)
- Scalabilità:

Il costo del calcolo parallelo

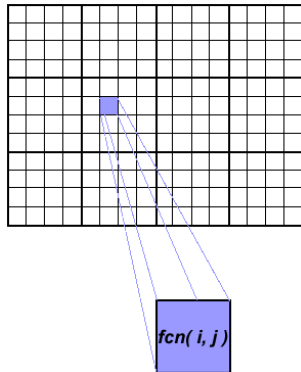
- Complessità:
 - progettazione, codifica, debugging, tuning, manutenzione
- Portabilità
 - migliorata con standard, ma ancora un problema (OSs, HW)
- Scalabilità:
 - un obiettivo con molti fattori in gioco (memoria, banda di rete, latenza, processori, librerie usate, etc.)

Presentazione

- 1 Progettazione di Programmi
 - Tecniche
 - Le problematiche
 - **Esempio 1: Array processing**
 - Esempio 2: Heat equation

Il problema: array processing

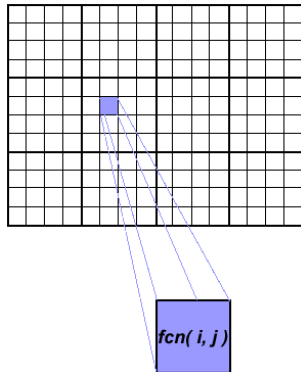
- Calcolo sull'elemento, indipendente dagli altri



Il problema: array processing

- Calcolo sull'elemento, indipendente dagli altri
- Il codice sequenziale sarebbe:

```
do j = 1,n  
do i = 1,n  
  a(i,j) = fcn(i,j)  
end do  
end do
```

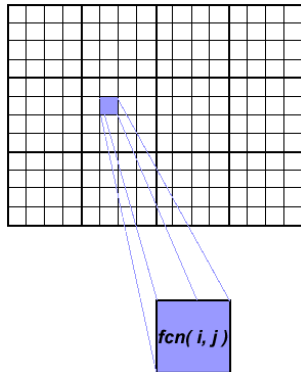


Il problema: array processing

- Calcolo sull'elemento, indipendente dagli altri
- Il codice sequenziale sarebbe:

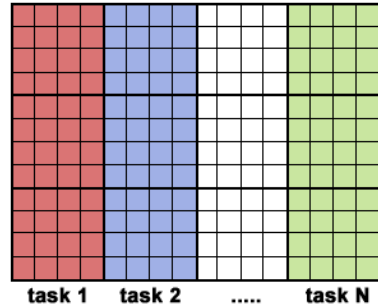
```
do j = 1,n  
do i = 1,n  
  a(i,j) = fcn(i,j)  
end do  
end do
```

- Embarassingly parallel



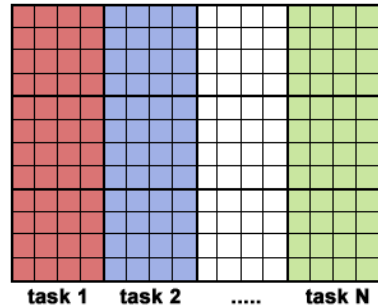
Array processing: soluzione 1

- Ogni processore ha una porzione



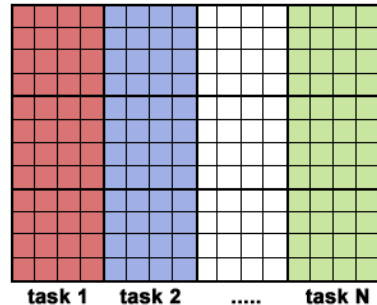
Array processing: soluzione 1

- Ogni processore ha una porzione
- Calcolo indipendente = nessuna comunicazione



Array processing: soluzione 1

- Ogni processore ha una porzione
- Calcolo indipendente = nessuna comunicazione
- Scelta del partizionamento unicamente influenzato da efficienza della cache



Soluzione 1 con SPMD

Modello Single Program Multiple Data (simile a SIMD)

```
find out if I am MASTER or WORKER

if I am MASTER

    initialize the array
    send each WORKER info on part of array it owns
    send each WORKER its portion of initial array

    receive from each WORKER results

else if I am WORKER
    receive from MASTER info on part of array I own
    receive from MASTER my portion of initial array

    # calculate my portion of array
    do j = my first column, my last column
        do i = 1, n
            a(i,j) = fcn(i,j)
        end do
    end do

    send MASTER results

endif
```



Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico

Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico
 - non efficiente con processori eterogenei

Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico
 - non efficiente con processori eterogenei
 - se la funzione da calcolare é estremamente veloce da calcolare per certi input (ad esempio, 0) e particolarmente onerosa in altri, allora la distribuzione in pezzi di ugual dimensione può non bilanciare il carico

Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico
 - non efficiente con processori eterogenei
 - se la funzione da calcolare é estremamente veloce da calcolare per certi input (ad esempio, 0) e particolarmente onerosa in altri, allora la distribuzione in pezzi di ugual dimensione può non bilanciare il carico
- Miglioramento con il pool di task

Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico
 - non efficiente con processori eterogenei
 - se la funzione da calcolare é estremamente veloce da calcolare per certi input (ad esempio, 0) e particolarmente onerosa in altri, allora la distribuzione in pezzi di ugual dimensione può non bilanciare il carico
- Miglioramento con il pool di task
- Processo Master: mantiene il pool di task e li distribuisce su richiesta ai worker, ottenendo (e assemblando) i risultati

Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico
 - non efficiente con processori eterogenei
 - se la funzione da calcolare é estremamente veloce da calcolare per certi input (ad esempio, 0) e particolarmente onerosa in altri, allora la distribuzione in pezzi di ugual dimensione può non bilanciare il carico
- Miglioramento con il pool di task
- Processo Master: mantiene il pool di task e li distribuisce su richiesta ai worker, ottenendo (e assemblando) i risultati
- Processo Worker: ripete: chiedi un task, esegilo e invia risultati al master

Array processing: soluzione 2

- Limiti della soluzione statica per il bilanciamento del carico
 - non efficiente con processori eterogenei
 - se la funzione da calcolare é estremamente veloce da calcolare per certi input (ad esempio, 0) e particolarmente onerosa in altri, allora la distribuzione in pezzi di ugual dimensione può non bilanciare il carico
- Miglioramento con il pool di task
- Processo Master: mantiene il pool di task e li distribuisce su richiesta ai worker, ottenendo (e assemblando) i risultati
- Processo Worker: ripete: chiedi un task, esegilo e invia risultati al master
- Bilanciamento dinamico del carico (dimensionamento del task (=granularità) critico per le prestazioni)

Soluzione 2

```
find out if I am MASTER or WORKER

if I am MASTER

    do until no more jobs
        send to WORKER next job
        receive results from WORKER
    end do

    tell WORKER no more jobs
else if I am WORKER

    do until no more jobs
        receive from MASTER next job

        calculate array element:  $a(i,j) = fcn(i,j)$ 

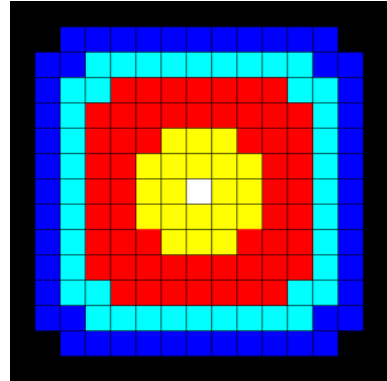
        send results to MASTER
    end do
endif
```

Presentazione

- 1 Progettazione di Programmi
 - Tecniche
 - Le problematiche
 - Esempio 1: Array processing
 - **Esempio 2: Heat equation**

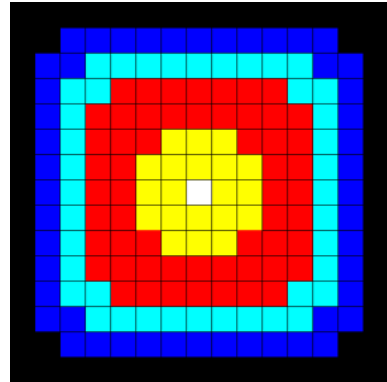
Il problema - 1

- La equazione descrive il cambiamento della temperature nel tempo



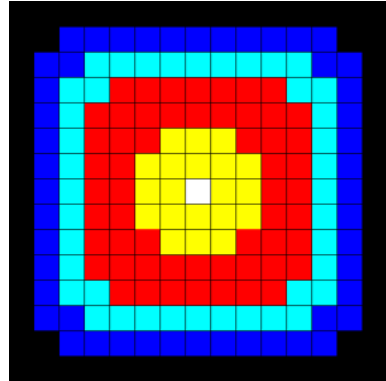
Il problema - 1

- La equazione descrive il cambiamento della temperature nel tempo
- Alta al centro e zero all'esterno, all'inizio



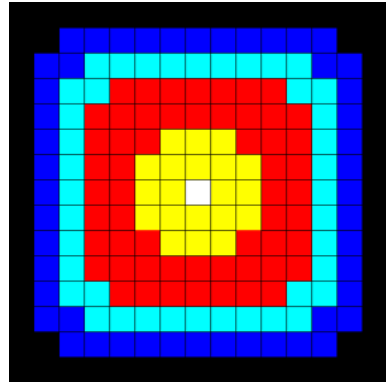
Il problema - 1

- La equazione descrive il cambiamento della temperature nel tempo
- Alta al centro e zero all'esterno, all'inizio
- Nel tempo, calcolare il cambiamento



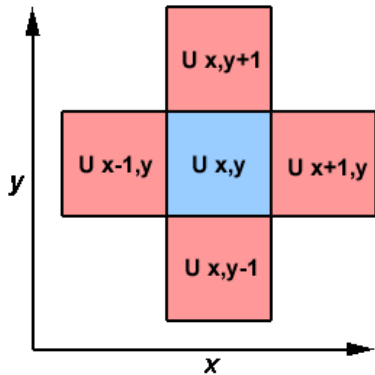
Il problema - 1

- La equazione descrive il cambiamento della temperature nel tempo
- Alta al centro e zero all'esterno, all'inizio
- Nel tempo, calcolare il cambiamento
- Problema che richiede comunicazione tra task



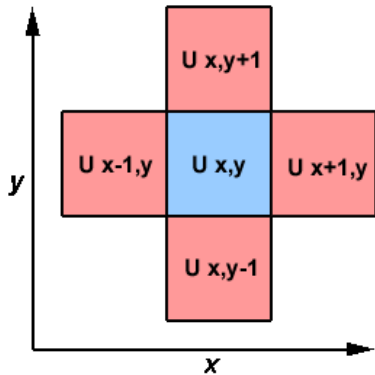
Il problema - 2

- Calcolo basato sul valore dei vicini



Il problema - 2

- Calcolo basato sul valore dei vicini
- $U_{x,y} = U_{x,y} + C_x \cdot (U_{x+1,y} + U_{x-1,y} - 2 \cdot U_{x,y}) + C_y \cdot (U_{x,y+1} + U_{x,y-1} - 2 \cdot U_{x,y})$



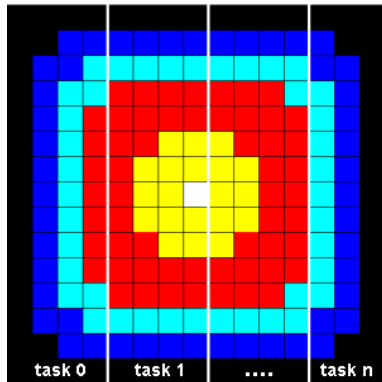
Codice seriale

Il codice seriale apparirebbe così:

```
do iy = 2, ny - 1
do ix = 2, nx - 1
  u2(ix, iy) =
    u1(ix, iy) +
      cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2.*u1(ix,iy)) +
      cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2.*u1(ix,iy))
end do
end do
```

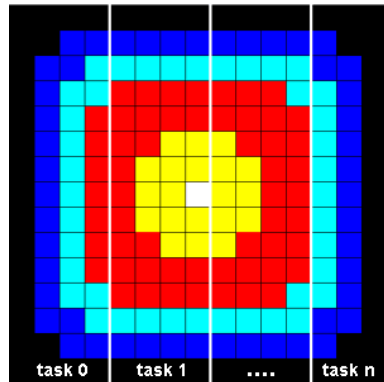
Soluzione 1

- Soluzione SPMD



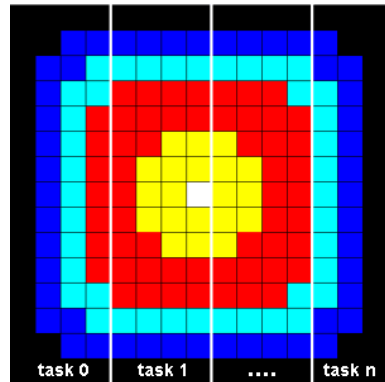
Soluzione 1

- Soluzione SPMD
- Array partizionato



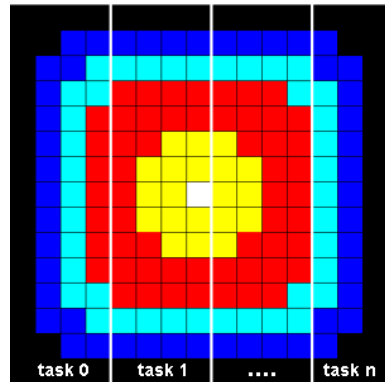
Soluzione 1

- Soluzione SPMD
- Array partizionato
- Elementi interni non dipendono da altri task



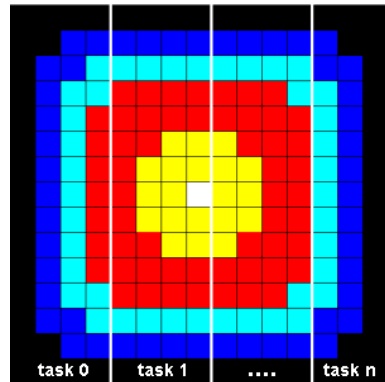
Soluzione 1

- Soluzione SPMD
- Array partizionato
- Elementi interni non dipendono da altri task
- Elementi al confine hanno bisogno di comunicazione



Soluzione 1

- Soluzione SPMD
- Array partizionato
- Elementi interni non dipendono da altri task
- Elementi al confine hanno bisogno di comunicazione
- Master-worker



```
find out if I am MASTER or WORKER

if I am MASTER
    initialize array
    send each WORKER starting info and subarray

    do until all WORKERS converge
        gather from all WORKERS convergence data
        broadcast to all WORKERS convergence signal
    end do

    receive results from each WORKER

else if I am WORKER
    receive from MASTER starting info and subarray

    do until solution converged
        update time
        send neighbors my border info
        receive from neighbors their border info

        update my portion of solution array

        determine if my solution has converged
        send MASTER convergence data
        receive from MASTER convergence signal
    end do

    send MASTER results

endif
```

● Soluzione SPMD

```
find out if I am MASTER or WORKER

if I am MASTER
    initialize array
    send each WORKER starting info and subarray

    do until all WORKERS converge
        gather from all WORKERS convergence data
        broadcast to all WORKERS convergence signal
    end do

    receive results from each WORKER

else if I am WORKER
    receive from MASTER starting info and subarray

    do until solution converged
        update time
        send neighbors my border info
        receive from neighbors their border info

        update my portion of solution array

        determine if my solution has converged
        send MASTER convergence data
        receive from MASTER convergence signal
    end do

    send MASTER results

endif
```

- Soluzione SPMD
- Convergenza indicata da stabilità nella soluzione

```
find out if I am MASTER or WORKER

if I am MASTER
    initialize array
    send each WORKER starting info and subarray

    do until all WORKERS converge
        gather from all WORKERS convergence data
        broadcast to all WORKERS convergence signal
    end do

    receive results from each WORKER

else if I am WORKER
    receive from MASTER starting info and subarray

    do until solution converged
        update time
        send neighbors my border info
        receive from neighbors their border info

        update my portion of solution array

        determine if my solution has converged
        send MASTER convergence data
        receive from MASTER convergence signal
    end do

    send MASTER results

endif
```

- Soluzione SPMD
- Convergenza indicata da stabilità nella soluzione
- Nel ciclo: send, receive, update array...

Critiche alla soluzione 1

- Comunicazione blocking: collo di bottiglia

Critiche alla soluzione 1

- Comunicazione blocking: collo di bottiglia
- Si può migliorare comunicazione non-blocking

Critiche alla soluzione 1

- Comunicazione blocking: collo di bottiglia
- Si può migliorare comunicazione non-blocking
- L'idea: aggiornare l'interno della propria zona, mentre si attendono i dati del bordo.

Soluzione 2 - codice

```
find out if I am MASTER or WORKER
```

```
if I am MASTER
```

```
  initialize array
```

```
  send each WORKER starting info and subarray
```

```
  do until all WORKERS converge
```

```
    gather from all WORKERS convergence data
```

```
    broadcast to all WORKERS convergence signal
```

```
  end do
```

```
  receive results from each WORKER
```

```
else if I am WORKER
```

```
  receive from MASTER starting info and subarray
```

```
  do until solution converged
```

```
    update time
```

```
    non-blocking send neighbors my border info
```

```
    non-blocking receive neighbors border info
```

```
    update interior of my portion of solution array
```

```
    wait for non-blocking communication complete
```

```
    update border of my portion of solution array
```

```
  determine if my solution has converged
```

```
    send MASTER convergence data
```

```
    receive from MASTER convergence signal
```

```
  end do
```

```
  send MASTER results
```

```
endif
```

- La comunicazione non-blocking viene solo iniziata

Soluzione 2 - codice

```
find out if I am MASTER or WORKER

if I am MASTER
  initialize array
  send each WORKER starting info and subarray

  do until all WORKERS converge
    gather from all WORKERS convergence data
    broadcast to all WORKERS convergence signal
  end do

  receive results from each WORKER

else if I am WORKER
  receive from MASTER starting info and subarray

  do until solution converged
    update time

    non-blocking send neighbors my border info
    non-blocking receive neighbors border info

    update interior of my portion of solution array
    wait for non-blocking communication complete
    update border of my portion of solution array

    determine if my solution has converged
    send MASTER convergence data
    receive from MASTER convergence signal
  end do

  send MASTER results

endif
```

- La comunicazione non-blocking viene solo iniziata
- Poi si passa al calcolo dell'interno ...

Soluzione 2 - codice

```
find out if I am MASTER or WORKER
```

```
if I am MASTER
  initialize array
  send each WORKER starting info and subarray

  do until all WORKERS converge
    gather from all WORKERS convergence data
    broadcast to all WORKERS convergence signal
  end do

  receive results from each WORKER

else if I am WORKER
  receive from MASTER starting info and subarray

  do until solution converged
    update time

    non-blocking send neighbors my border info
    non-blocking receive neighbors border info

    update interior of my portion of solution array
    wait for non-blocking communication complete
    update border of my portion of solution array

    determine if my solution has converged
    send MASTER convergence data
    receive from MASTER convergence signal
  end do

  send MASTER results

endif
```



- La comunicazione non-blocking viene solo iniziata
- Poi si passa al calcolo dell'interno ...
- ... e si termina quando si ricevono i dati dall'esterno

