Capitolo 3

Il paradigma dello scambio di messaggi (message passing)

Il modello di programmazione per calcolatori monoprocessori è ormai standard. L'algoritmo viene progettato per essere eseguito da un singolo processore che può accedere alla memoria eseguendo una istruzione dopo l'altra.

Il *"message passing"* è un modello per la progettazione di algoritmi in un ambiente di calcolo parallelo per calcolatori MIMD a memoria distribuita. Nel progettare l'algoritmo, in questo caso, si deve tenere conto che esistono piú processori, ognuno con una *"propria"* memoria, capaci di eseguire ciascuno un *"proprio"* insieme di istruzioni. Chiaramente, i processori devono sincronizzarsi per risolvere un medesimo problema, e ciò avviene attraverso lo scambio di messaggi. Il concetto fondamentale alla base del paradigma *message-passing* è che i processori[1] comunicano tra loro inviando e ricevendo dati.

Nel prossimo paragrafo illustreremo alcuni algoritmi di calcolo

[1] Con il termine *"processo"* si indica un qualunque programma in esecuzione. Con il termine *"processore"* si indica l'hardware che esegue il processo. Un processore può eseguire piú processi. Nel seguito si supporrà che su ciascun processore venga eseguito un singolo processo. Quindi i termini processore e processo verranno utilizzati anche come sinonimi.

matriciale mettendo in evidenza in ciascuno la strategia alla base dell'introduzione del parallelismo.

3.1 L'operazione matrice per vettore

Problema

Calcolo del prodotto:

$$Ax = y \quad , \; A \in \Re^{n \times n}, \;\; x, y \in \Re^n$$

su un calcolatore MIMD a memoria distribuita con p processori, con A matrice quadrata di ordine n ed x ed y vettori di dimensione n.

L'algoritmo sequenziale piú naturale é quello che prevede il calcolo del vettore y componente per componente:

$$y_i = \sum_{j=1}^{n} a_{i,j} \cdot x_j, \;\; con \;\; i = 1, \ldots, n$$

effettuando i prodotti scalari di ciascuna riga di A per il vettore x, viene cosí calcolata una componente per volta secondo un ordine prestabilito[2]. Supponendo di seguire l'ordine naturale delle componenti y_i di y, per $i = 0, \ldots, n-1$, viene effettuato il prodotto della prima riga di A per il vettore x, ottenendo y_0, la prima componente di y. Viene poi moltiplicata la seconda riga di A per il vettore x ottenendo cosí la seconda componente, y_1, di y, e cosí via fino a calcolare tutte le n componenti del vettore y. L'algoritmo è descritto nella *Procedura* 3.1 .

[2] A titolo di esempio è stata presa in considerazione la versione *"per righe"* dell'algoritmo matrice-vettore perché implementa l'usuale definizione *"righe per colonne"* del prodotto di una matrice per un vettore, anche se questa risulta essere non sempre la piú efficiente.

```
procedure matvet(A, x, n, y)
integer n,i
real A(n,n),x(n),y(n)
for i = 0 to n − 1 do
    % inizializzazione a zero della componente
    % i-esima del vettore y
    y_i := 0
    % prodotto della riga i-esima della matrica A
    % per il vettore x
    for j = 0 to n − 1 do
        y_i := y_i + A_ij x_j
    endfor
endfor
return
end matvet
```

Procedura 3.1 - Algoritmo sequenziale per il calcolo del prodotto matrice-vettore (versione righe per colonne).

Osserviamo che il calcolo di ciascun prodotto delle righe della matrice A per il vettore x può essere effettuato in modo indipendente dagli altri prodotti tra le altre righe di A e il vettore x.

Per eseguire la stessa operazione in un ambiente MIMD a memoria distribuita, l'idea piú naturale è quella di distribuire il calcolo delle componenti di y, ovvero di distribuire il calcolo degli n prodotti scalari tra le righe di A e il vettore y.

Supponiamo che A sia una matrice quadrata di ordine 6, x e y vettori di lunghezza 6. Supponiamo inoltre di risolvere il problema su $p = 2$ processori. Le operazioni da effettuare sono le seguenti:

1) Distribuzione dei dati.

Disponendo di 2 processori la decomposizione piú naturale è quella in cui si divide la matrice A in 2 blocchi di righe: il primo costituito

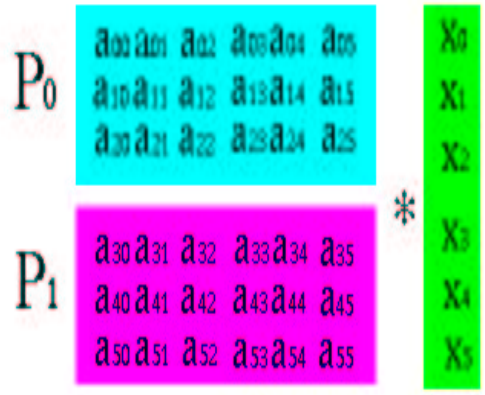


Figura 3.1: *Distribuzione dei dati: ad ogni processore viene assegnato un blocco di righe della matrice A e tutto il vettore x.*

dalle prime tre righe e il secondo costituito dalle seconde tre righe di A [3]. Il vettore x verrà assegnato invece ad entrambi i processori (tale distribuzione è rappresentata in Fig. 3.1).

Notiamo che, avendo supposto di avere 2 processori con 2 memorie locali, i dati distribuiti vengono memorizzati localmente nella memoria di ciascun processore (Fig. 3.2).

Introducendo una notazione algoritmica[4], i dati saranno cosí distribuiti tra i processori:

[3] Analogamente, si può pensare di utilizzare una distribuzione ciclica delle righe di A ai 2 processori; piú precisamente la prima riga a P_0, la seconda a P_1, la terza a P_0, la quarta a P_1 e cosí continuando.

[4] Quì e nel seguito si userà la notazione, tipica, ad esempio, del MATLAB, per indicare una matrice e/o una parte di questa. In particolare, la notazione $A(i:j,:)$ indica che si stanno considerando dalla (i+1)-esima alla (j+1)-esima riga e tutte le colonne della matrice A; la notazione $A(:, i:j)$ indica che si stanno considerando tutte le righe della matrice A e le colonne dalla (i+1)-esima alla (j+1)-esima.

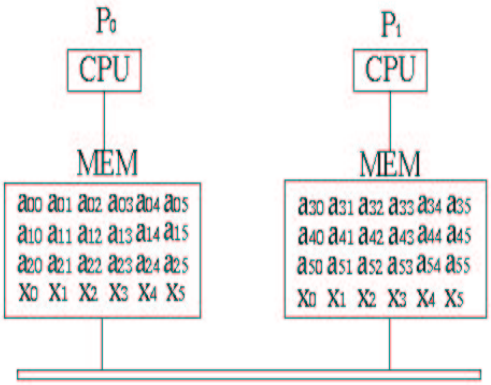


Figura 3.2: *Distribuzione degli elementi della matrice A e del vettore x tra i processori. Al processore P_0 vengono assegnate le prime tre righe di A e tutto il vettore x, al processore P_1 vengono assegnate le seconde tre righe di A e tutto il vettore x.*

Algoritmo processore P_0	**Algoritmo processore P_1**
$A_{loc}(0:2,:) :=$A(0:2,:)	$A_{loc}(3:5,:) :=$A(3:5,:)
$x_{loc} := x(0:5)$	$x_{loc} := x(0:5)$
$y_{loc} := y(0:5)$	$y_{loc} := y(0:5)$

A_{loc}, x_{loc} ed y_{loc} sono variabili locali residenti nella memoria di ciascun processore. È importante ricordare che ogni nodo non ha accesso alle variabili locali di un altro nodo e che ognuno dei due processori calcola tre delle componenti del vettore soluzione y.

2) Calcolo dei prodotti parziali.

Ciascun processore effettua il prodotto delle righe di A righe per il vettore y utilizzando i dati residenti nella propria memoria:

Algoritmo processore P_0

$$y_{loc}(0) := a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + a_{03}x_3 + a_{04}x_4 + a_{05}x_5$$
$$y_{loc}(1) := a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}x_5$$
$$y_{loc}(2) := a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + a_{25}x_5$$

Algoritmo processore P_1

$$y_{loc}(3) := a_{30}x_0 + a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 + a_{35}x_5$$
$$y_{loc}(4) := a_{40}x_0 + a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 + a_{45}x_5$$
$$y_{loc}(5) := a_{50}x_0 + a_{51}x_1 + a_{52}x_2 + a_{53}x_3 + a_{54}x_4 + a_{55}x_5$$

I processori eseguono lo stesso algoritmo su dati diversi, secondo lo schema mostrato nella *Procedura* 3.2 .

Algoritmo processore P_0

```
⋮
for i=0 to 2 do
   y_loc(i) := 0
   for j = 0 to 5 do
      y_loc(i) := y_loc(i) + A_loc(i, j)x_loc(j)
   endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i=3 to 5 do
   y_loc(i) := 0
   for j = 0 to 5 do
      y_loc(i) := y_loc(i) + A_loc(i, j)x_loc(j)
   endfor
endfor
⋮
```

Procedura 3.2 - Algoritmo per il calcolo delle componenti di y - I Strategia

3) Combinazione dei risultati.

Terminata la fase di calcolo, il processore P_0 ha le prime 3 componenti del vettore soluzione, mentre, il processore P_1 ha le ultime 3. Se richiediamo che ambedue i processori abbiano tutte le componenti del vettore y, dobbiamo considerare uno scambio dei dati tra

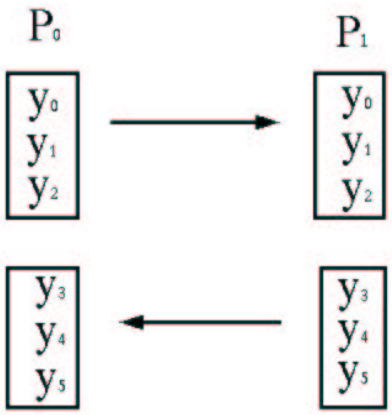


Figura 3.3: *In figura è rappresentato lo scambio delle componenti del vettore y tra i due processori.*

i 2 processori. L'operazione consta di un solo passo:

il processore P_0 invia al processore P_1 la prima, la seconda e la terza componente del vettore y e il processore P_1 invia al processore P_0 la quarta, la quinta e la sesta componente del vettore y (Fig 3.3). In questo modo entrambi i processori avranno a disposizione tutto il vettore soluzione.
Si hanno, cosí, le operazioni di comunicazione seguenti:

Algoritmo processore P_0	**Algoritmo processore** P_1
send($y_{loc}(0:2)$,1)	send($y_{loc}(3:5)$,0)
recv($y_{loc}(3:5)$,1)	recv($y_{loc}(0:2)$,0)

La funzione *send* ha come argomenti gli elementi da inviare e l'identificativo del processore a cui inviarli. La funzione *recv* ha come argomenti gli elementi da ricevere e l'identificativo del processore da cui riceverli. Gli algoritmi per i due processori sono riportati nella *Procedura* 3.3 .

I STRATEGIA

procedure MATVET (proc. 0)	**procedure MATVET (proc. 1)**
begin	**begin**
% distribuzione dei dati	% distribuzione dei dati
$A_{loc} := A(0:2,:)$	$A_{loc} := A(3:5,:)$
$x_{loc} := x(0:5)$	$x_{loc} := x(0:5)$
$y_{loc} := y(0:5)$	$y_{loc} := y(0:5)$
% calcolo di alcune componenti	% calcolo di alcune componenti
% del vettore soluzione	% del vettore soluzione
for i=0 to 2 **do**	**for** i=3 to 5 **do**
$y_{loc}(i) := 0$	$y_{loc}(i) := 0$
for $j = 0$ **to** 5 **do**	**for** $j = 0$ **to** 5 **do**
$y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$	$y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$
endfor	**endfor**
endfor	**endfor**
% scambio dei risultati	% scambio dei risultati
send$(y_{loc}(0:2), 1)$	**recv**$(y_{loc}(0:2), 0)$
recv$(y_{loc}(3:5), 1)$	**send**$(y_{loc}(3:5), 0)$
end	**end**

Procedura 3.3 - Algoritmo per il calcolo del prodotto matrice-vettore - I Strategia.

Osserviamo che l'aver introdotto il parallelismo nel calcolo delle componenti di y, ovvero l'aver distribuito tra i 2 processori il calcolo di prodotti scalari delle righe di A per il vettore y, si riflette nell'algoritmo in una decomposizione del primo ciclo iterativo "*for ... end for*"; per il processore P_0 il ciclo su i procede da 0 a 2 mentre per il processore P_1 da 3 a 5. In questo modo il processore P_0 calcola i primi 3 prodotti scalari mentre il processore P_1 gli altri 3.

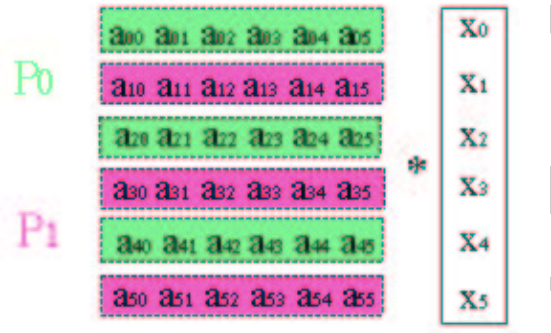


Figura 3.4: *Distribuzione dei dati: ad ogni processore vengono assegnate ciclicamente le righe della matrice A e tutto il vettore x.*

♣ Esempio 9

Eseguiamo il prodotto

$$A \cdot x = y$$

con $A \in \Re^{6\times 6}$, x e $y \in \Re^6$.

1) Distribuzione dei dati.

Invece di suddividere la matrice in 2 blocchi di righe e assegnare il primo blocco al processore P_0 ed il secondo al processore P_1, assegnamo la prima, la terza e la quinta riga della matrice A al processore P_0, le righe rimanenti al processore P_1. Questo tipo di distribuzione è detta *ciclica per righe* (Scattered). Il vettore x verrà assegnato ad entrambi i processori (tale distribuzione è rappresentata in Fig. 3.4).

I dati cosí distribuiti vengono memorizzati localmente nella memoria di ciascuno dei due processori; tale schema è mostrato graficamente in Fig. 3.5.

Introducendo una notazione algoritmica, i dati saranno cosí distribuiti tra i processori:

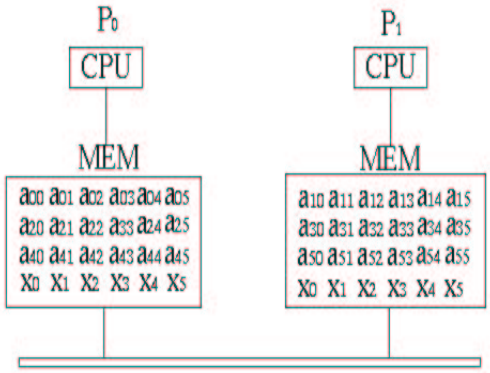


Figura 3.5: *Distribuzione degli elementi della matrice A e del vettore x tra i processori. Al processore P_0 vengono essagnate le righe di indice pari della matrice A e tutto il vettore x; al processore P_1 vengono assegnate le righe di indice dispari della matrice A e tutto il vettore x.*

Algoritmo processore P_0
$A_{loc}(0:2:5,:) :=$A(0:2:5,:)
$x_{loc} := x(0:5)$
$y_{loc} := y(0:5)$

Algoritmo processore P_1
$A_{loc}(1:2:5,:) :=$A(1:2:5,:)
$x_{loc} := x(0:5)$
$y_{loc} := y(0:5)$

A_{loc}, x_{loc} ed y_{loc} sono variabili locali residenti nella memoria di ciascun processore. Con tale distribuzione dei dati ogni processore calcola tre delle componenti del vettore soluzione y.

2) Calcolo dei prodotti parziali.

Ciascun processore effettua il prodotto righe per colonne con i dati che possiede nella propria memoria:

Algoritmo processore P_0
$y_{loc}(0) := a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + a_{03}x_3 + a_{04}x_4 + a_{05}x_5$
$y_{loc}(2) := a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + a_{25}x_5$
$y_{loc}(4) := a_{40}x_0 + a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 + a_{45}x_5$

Algoritmo processore P_1

$y_{loc}(1) := a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}x_5$
$y_{loc}(3) := a_{30}x_0 + a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 + a_{35}x_5$
$y_{loc}(5) := a_{50}x_0 + a_{51}x_1 + a_{52}x_2 + a_{53}x_3 + a_{54}x_4 + a_{55}x_5$

I processori eseguono lo stesso algoritmo su dati diversi, secondo lo schema mostrato nella *Procedura* 3.4 .

Algoritmo processore P_0

```
⋮
for i=0 to 5 step 2 do
    y_loc(i) := 0
    for j = 0 to 5 do
        y_loc(i) := y_loc(i) + A_loc(i,j)x_loc(j)
    endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i=1 to 5 step 2 do
    y_loc(i) := 0
    for j = 0 to 5 do
        y_loc(i) := y_loc(i) + A_loc(i,j)x_loc(j)
    endfor
endfor
⋮
```

Procedura 3.4 - Algoritmo per il calcolo delle componenti di y relative ai dati assegnati - Distribuzione ciclica delle righe della matrice A.

3) Combinazione dei risultati.
Terminata la fase di calcolo il processore P_0 ha le componenti 0, 2 e 4 del vettore soluzione, mentre, il processore P_1 ha le componenti 1, 3 e 5. Quindi, deve esserci uno scambio dei dati tra i processori in modo che abbiano ambedue tutte le componenti del vettore soluzione y. L'operazione consta di un solo passo:

Il processore P_0 invia al processore P_1 la prima, la terza e la quinta componente del vettore y e il processore P_1 invia al processore P_0 la seconda, la quarta e la sesta componente del vettore y (Fig. 3.6). In questo modo entrambi i processori avranno a disposizione tutto il vettore soluzione.
Si hanno, cosí, le operazioni di comunicazione seguenti:

Algoritmo processore P_0

send($y_{loc}(0:2:5)$,1)
recv($y_{loc}(1:2:5)$,1)

Algoritmo processore P_1

send($y_{loc}(1:2:5)$,0)
recv($y_{loc}(0:2:5)$,0)

Gli algoritmi per i due processori sono riportati nella *Procedura* 3.5 .

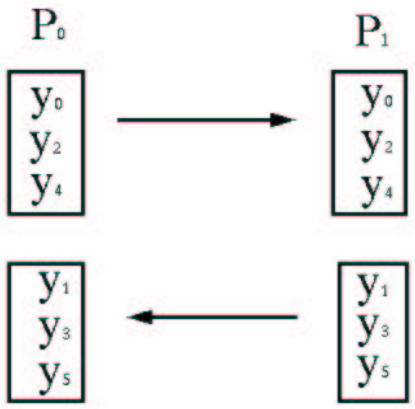


Figura 3.6: *In figura è rappresentato lo scambio delle componenti del vettore y tra i due processori.*

Distribuzione ciclica delle righe di A

procedure MATVET (proc. 0)

```
begin
% distribuzione dei dati
   A_loc := A(0:2:5,:)
   x_loc := x(0:5)
   y_loc := y(0:5)
% calcolo di alcune componenti
% del vettore soluzione
   for i=0 to 5 step 2 do
     y_loc(i) := 0
     for j = 0 to 5 do
       y_loc(i) := y_loc(i) + A_loc(i,j)x_loc(j)
     endfor
   endfor
% scambio dei risultati
   send(y_loc(0:2:5),1)
   recv(y_loc(1:2:5),1)
end
```

procedure MATVET (proc. 1)

```
begin
% distribuzione dei dati
   A_loc := A(1:2:5,:)
   x_loc := x(0:5)
   y_loc := y(0:5)
% calcolo di alcune componenti
% del vettore soluzione
   for i=1 to 5 step 2 do
     y_loc(i) := 0
     for j = 0 to 5 do
       y_loc(i) := y_loc(i) + A_loc(i,j)x_loc(j)
     endfor
   endfor
% scambio dei risultati
   recv(y_loc(0:2:5),0)
   send(y_loc(1:2:5),0)
end
```

Procedura 3.5 - Algoritimo per il calcolo del prodotto matrice per vettore - la matrice A è distribuita ciclicamente lungo le righe.

△

La distribuzione dei dati utilizzata nell'esempio 9 è un caso particolare della *Distribuzione Ciclica a Blocchi di Righe*[5] in cui le righe della matrice A sono suddivise in blocchi di dimensione nb e poi, ciascun blocco viene assegnato ad un processore, in modo che la k-esima riga di A appartenga al processore il cui identificativo è:

$$id = mod(\lfloor k/nb \rfloor, p)$$

dove p indica il numero di processori ed il simbolo $\lfloor x \rfloor$ la parte intera di x. Nell'esempio 9 $nb = 1$ e $p = 2$.

Una strategia diversa di distribuzione della matrice A si basa sulla decomposizione di A in blocchi di colonne[6]. In tal caso le operazioni da effettuare sono le seguenti:

1) Distribuzione dei dati.
Dividiamo la matrice A in 2 blocchi di colonne: il primo costituito dalle prime tre colonne e il secondo costituito dalle restanti tre colonne. Dividiamo il vettore x in blocchi di righe (Fig. 3.7). In un linguaggio algoritmico, introducendo le variabili locali A_{loc}, x_{loc} e y_{loc}, residenti nella memoria di ciascun nodo, si ha:

Algoritmo processore P_0
$A_{loc}(:, 0:2) := A(:, 0:2)$
$x_{loc} := x(0:2)$
$y_{loc} := y(0:5)$

Algoritmo processore P_1
$A_{loc}(:, 3:5) := A(:, 3:5)$
$x_{loc}(3:5) := x(3:5)$
$y_{loc} := y(0:5)$

Tale suddivisione è rappresentata graficamente in Fig. 3.8.

[5]Tale distribuzione è utilizzata nelle librerie matematiche Blas2 e Blas3 allo scopo di sfruttare al meglio la struttura della matrice A e rendere gli algoritmi ben bilanciati.

[6]Analogamente, anche in questo caso, si può adottare una distribuzione ciclica delle colonne delle matrici; piú precisamente la prima colonna a P_0, la seconda a P_1, la terza a P_0, la quarta a P_1 e cosí continuando.

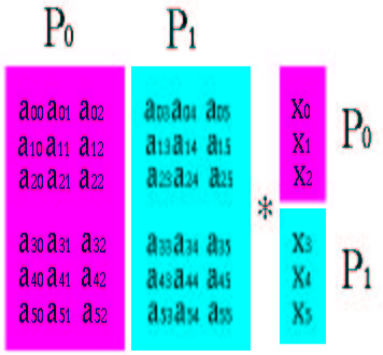


Figura 3.7: *Distribuzione dei dati nella II strategia: ad ogni processore viene assegnato un blocco di colonne della matrice A e un blocco di righe del vettore x.*

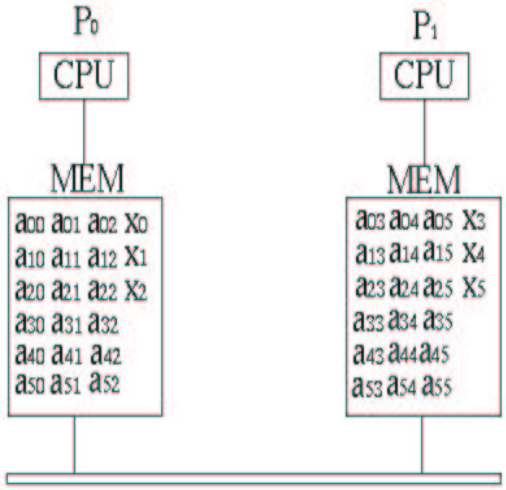


Figura 3.8: *Distribuzione degli elementi della matrice A e del vettore x tra i processori. Al processore P_0 verranno assegnate le prime tre colonne di A e le prime tre componenti di x, al processore P_1 verranno assegnate le seconde tre colonne di A e le seconde tre componenti di x.*

2) Calcolo dei prodotti parziali.
Ciascun processore effettua il prodotto righe per colonne con i dati che ha nella propria memoria:

Alagoritmo processore P_0
$y_{loc}(0) := a_{00}x_0 + a_{01}x_1 + a_{02}x_2$
$y_{loc}(1) := a_{10}x_0 + a_{11}x_1 + a_{12}x_2$
$y_{loc}(2) := a_{20}x_0 + a_{21}x_1 + a_{22}x_2$
$y_{loc}(3) := a_{30}x_0 + a_{31}x_1 + a_{32}x_2$
$y_{loc}(4) := a_{40}x_0 + a_{41}x_1 + a_{42}x_2$
$y_{loc}(5) := a_{50}x_0 + a_{51}x_1 + a_{52}x_2$

Alagoritmo processore P_1
$y_{loc}(0) := a_{03}x_3 + a_{04}x_4 + a_{05}x_5$
$y_{loc}(1) := a_{13}x_3 + a_{14}x_4 + a_{15}x_5$
$y_{loc}(2) := a_{23}x_3 + a_{24}x_4 + a_{25}x_5$
$y_{loc}(3) := a_{33}x_3 + a_{34}x_4 + a_{35}x_5$
$y_{loc}(4) := a_{43}x_3 + a_{44}x_4 + a_{45}x_5$
$y_{loc}(5) := a_{53}x_3 + a_{54}x_4 + a_{55}x_5$

I processori eseguono lo stesso algoritmo su dati diversi, secondo lo schema riportato nella *Procedura* 3.6 .

Algoritmo processore P_0	**Algoritmo processore P_1**
$\vdots$	$\vdots$
for $i = 0$ **to** 5 **do**	**for** $i = 0$ **to** 5 **do**
$y_{loc}(i) := 0$	$y_{loc}(i) := 0$
for j=0 to 2 **do**	**for** j=3 to 5 **do**
$y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$	$y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$
endfor	**endfor**
endfor	**endfor**
$\vdots$	$\vdots$

Procedura 3.6 - Algoritmo per il calcolo dei prodotti parziali- II Strategia

In questa seconda strategia, come si può osservare dall'algoritmo, l'aver distribuito il calcolo di ciascun prodotto scalare, ovvero l'aver decomposto la matrice A per colonne, si riflette nell'algoritmo nella decomposizione del secondo ciclo "*for ... end for*"; per il processore P_0 il ciclo su j procede da 0 a 2, per il processore P_1 da 3 a 5.

In questo modo ciascun processore può calcolare somme parziali di ciascuna componente del vettore y.

3) Combinazione dei risultati.
Terminata la fase di calcolo, i processori devono collezionare i dati in modo da avere ambedue il risultato finale. Entrambi i processori, P_0 e P_1, hanno una somma parziale di tutte le componenti del vettore risultato. Per ottenere, quindi, il risultato finale i due processori devono scambiare tra loro le somme parziali ottenute e sommare le analoghe. L'operazione consta di tre passi:

I PASSO

Il processore P_0 invia al processore P_1 la seconda parte del vettore y e il processore P_1 invia al processore P_0 la prima parte del vettore y (Fig. 3.9). In questo modo entrambi i processori avranno a disposizione i dati di cui hanno bisogno per calcolare il vettore somma finale. Indicata con $aus(0:2)$ la variabile locale nella quale ogni processore memorizza la porzione di vettore inviatagli dall'altro, si hanno le operazioni di comunicazione seguenti:

Algoritmo processore P_0	**Algoritmo processore P_1**
$send(y_{loc}(3:5), 1)$ $recv(aus(0:2), 1)$	$send(y_{loc}(0:2), 0)$ $recv(aus(0:2), 0)$

II PASSO

Per far sì che ciascun processore abbia una parte del vettore soluzione y, il processore P_0 calcola la somma delle prime tre componenti del vettore y_{loc} e le tre componenti del vettore aus inviatogli dal processore P_1, il processore P_1 calcola la somma tra le ultime tre componenti del vettore y_{loc} e le tre componenti del vettore aus inviatogli dal processore P_0 (*Procedura* 3.7)

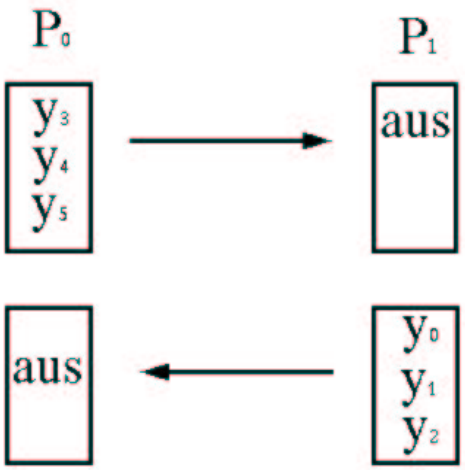


Figura 3.9: *Il processore P_0 invia al processore P_1 la seconda parte del vettore y e il processore P_1 invia al processore P_0 la prima parte del vettore y. Il processore che riceve memorizza il dato inviatogli nella variabile locale aus.*

Algoritmo processore P_0	**Algoritmo processore P_1**

⋮
for i=0 to 2 **do**
$y_{loc}(i) := y_{loc}(i) + aus(i)$
endfor
⋮
	⋮
	for i=3 to 5 **do**
	$y_{loc}(i) := y_{loc}(i) + aus(i-3)$
	endfor
	⋮

Procedura 3.7 - Calcolo delle componenti del vettore soluzione y nei due processori - II Strategia

III PASSO

Ciascun processore invia all'altro la propria parte del vettore y aggiornata al passo precedente (Fig. 3.10). In questo modo ogni processore avrà tutte le componenti del vettore prodotto y in y_{loc}.

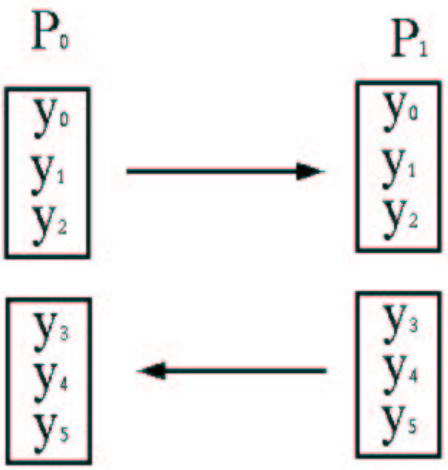


Figura 3.10: *Il processore P_0 invia al processore P_1 la prima parte del vettore risultante y e il processore P_1 invia al processore P_0 la seconda parte del vettore risultante y. Entrambi i processori avranno cosí l'intero vettore risultante y.*

Algoritmo processore P_0
$send(y_{loc}(0:2), 1)$
$recv(y_{loc}(3:5), 1)$

Algoritmo processore P_1
$send(y_{loc}(3:5), 0)$
$recv(y_{loc}(0:2), 0)$

Gli algoritmi per i due processori sono riportati nella *Procedura* 3.8.

II STRATEGIA

procedure MATVET proc. 0

```
begin
% distribuzione dei dati
  A_loc := A(:, 0 : 2)
  x_loc := x(0 : 2)
  y_loc := y(0 : 5)
% calcolo dei prodotti parziali
  for i = 0 to 5 do
    y_loc(i) := 0
    for j = 0 to 2 do
      y_loc(i) := y_loc(i) + A_loc(i, j)x_loc(j)
    endfor
  endfor
% combinazione dei risultati parziali
  send(y_loc(3 : 5), 1)
  recv(aus(0 : 2), 1)
  for i=0 to 2 do
    y_loc(i) := y_loc(i) + aus(i)
  endfor
% scambio delle componenti aggiornate
  send(y_loc(0 : 2), 1)
  recv(y_loc(3 : 5), 1)
end
```

procedure MATVET proc. 1

```
begin
% distribuzione dei dati
  A_loc := A(:, 3 : 5)
  x_loc := x(3 : 5)
  y_loc := y(0 : 5)
% calcolo dei prodotti parziali
  for i = 0 to 5 do
    y_loc(i) := 0
    for j = 3 to 5 do
      y_loc(i) := y_loc(i) + A_loc(i, j)x_loc(j)
    endfor
  endfor
% combinazione dei risultati parziali
  recv(aus(0 : 2), 0)
  send(y_loc(0 : 2), 0)
  for i=3 to 5 do
    y_loc(i) := y_loc(i) + aus(i − 3)
  endfor
% scambio delle componenti aggiornate
  recv(y_loc(0 : 2), 0)
  send(y_loc(3 : 5), 0)
end
```

Procedura 3.8 - Algoritmi per il calcolo del prodotto matrice-vettore - II Strategia

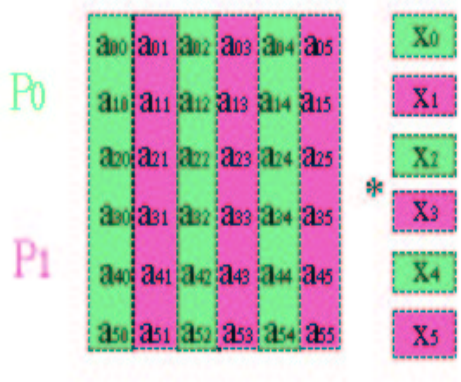


Figura 3.11: *Distribuzione dei dati: ad ogni processore vengono assegnate ciclicamente le colonne della matrice A e le righe del vettore x.*

♣ Esempio 10

Eseguiamo il prodotto

$$A \cdot x = y$$

con $A \in \Re^{6\times 6}$, x e $y \in \Re^6$. Le operazioni da eseguire sono le seguenti:

1) Distribuzione dei dati.

Assegnamo la prima, la terza e la quinta colonna della matrice A al processore P_0, le rimanenti colonne al processore P_1. Questo tipo di distribuzione è denominata *ciclica per colonne* (Scattered). Le componenti di indice pari del vettore x veranno assegnate al processore P_0, quelle di indice dispari al processore P_1 (tale distribuzione è rappresentata in Fig. 3.11).

I dati cosí distribuiti vengono memorizzati localmente nella memoria di ciascuno dei due processori; tale schema è mostrato graficamente in Fig. 3.12.

Introducendo una notazione algoritmica, i dati saranno distribuiti tra i processori nel modo seguente:

Algoritmo processore P_0
$A_{loc}(:, 0:2:5) := A(:, 0:2:5)$ $x_{loc}(0:2:5) := x(0:2:5)$ $y_{loc} := y(0:5)$

Algoritmo processore P_1
$A_{loc}(:, 1:2:5) := A(:, 1:2:5)$ $x_{loc}(1:2:5) := x(1:2:5)$ $y_{loc} := y(0:5)$

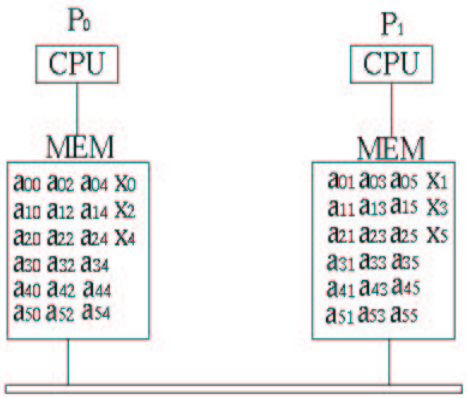


Figura 3.12: *Distribuzione degli elementi della matrice A e del vettore x tra i processori. Al processore P_0 vengono assegnate le colonne di indice pari della matrice A e le componenti di indice pari del vettore x; al processore P_1 vengono assegnate le colonne di indice dispari della matrice A e le componenti di indice dispari del vettore x.*

A_{loc}, x_{loc} ed y_{loc} sono variabili locali residenti nella memoria di ciascun processore. Con tale distribuzione dei dati ogni processore calcola tre delle componenti parziali del vettore soluzione y.

2) Calcolo dei prodotti parziali.
Ciascun processore effettua il prodotto righe per colonne con i dati che ha nella propria memoria:

Alagoritmo processore P_0
$y_{loc}(0) := a_{00}x_0 + a_{02}x_2 + a_{04}x_4$
$y_{loc}(1) := a_{10}x_0 + a_{12}x_2 + a_{14}x_4$
$y_{loc}(2) := a_{20}x_0 + a_{22}x_2 + a_{24}x_4$
$y_{loc}(3) := a_{30}x_0 + a_{32}x_2 + a_{34}x_4$
$y_{loc}(4) := a_{40}x_0 + a_{42}x_2 + a_{44}x_4$
$y_{loc}(5) := a_{50}x_0 + a_{52}x_2 + a_{54}x_4$

Alagoritmo processore P_1
$y_{loc}(0) := a_{01}x_1 + a_{03}x_3 + a_{05}x_5$
$y_{loc}(1) := a_{11}x_1 + a_{13}x_3 + a_{15}x_5$
$y_{loc}(2) := a_{21}x_1 + a_{23}x_3 + a_{25}x_5$
$y_{loc}(3) := a_{31}x_1 + a_{33}x_3 + a_{35}x_5$
$y_{loc}(4) := a_{41}x_1 + a_{43}x_3 + a_{45}x_5$
$y_{loc}(5) := a_{51}x_1 + a_{53}x_3 + a_{55}x_5$

I processori eseguono lo stesso algoritmo su dati diversi, secondo lo schema riportato nella *Procedura* 3.9.

Algoritmo processore P_0	**Algoritmo processore P_1**

$\vdots$
for $i = 0$ **to** 5 **do**
  $y_{loc}(i) := 0$
  for j=0 to 5 step 2 **do**
  $y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$
  endfor
endfor
$\vdots$
	$\vdots$
	for $i = 0$ **to** 5 **do**
	  $y_{loc}(i) := 0$
	  for j=1 to 5 step 2 **do**
	  $y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$
	  endfor
	endfor
	$\vdots$

Procedura 3.9 - Algoritmo per il calcolo dei prodotti parziali - Distribuzione ciclica per colonne.

In questa seconda strategia, come si può osservare dall'algoritmo, il parallelismo è stato introdotto nel secondo ciclo "*for ... end for*"; per il processore P_0 il ciclo su j procede da 0 a 5 di passo 2, per il processore P_1 da 1 a 5 di passo 2. In questo modo ciascun processore può calcolare somme parziali di ciascuna componente del vettore y.

3) Combinazione dei risultati.
Terminata la fase di calcolo, i processori devono collezionare i dati in modo da avere ambedue il risultato finale. Entrambi i processori, P_0 e P_1, hanno una somma parziale di tutte le componenti del vettore risultato. Per ottenere quindi il risultato finale i due processori devono scambiare tra loro le somme parziali ottenute e sommare le corrispondenti. L'operazione consta di tre passi che coincidono con i tre passi analizzati al punto 3) nella II Strategia.

Gli algoritmi per i due processori sono riportati nella *Procedura* 3.10 .

Distribuzione Ciclica per Colonne	
procedure MATVET proc. 0	**procedure MATVET proc. 1**

begin
% distribuzione dei dati
  $A_{loc}(:, 0:2:5) := A(:, 0:2:5)$
  $x_{loc}(0:2:5) := x(0:2:5)$
  $y_{loc} := y(0:5)$
	begin
	% distribuzione dei dati
	  $A_{loc}(:, 1:2:5) := A(:, 1:2:5)$
	  $x_{loc}(1:2:5) := x(1:2:5)$
	  $y_{loc} := y(0:5)$

Procedura 3.10 - Algoritmi per il calcolo del prodotto matrice-vettore - Distribuzione ciclica per colonne della matrice A - continua

```
% calcolo dei prodotti parziali
  for i = 0 to 5 do
    y_loc(i) := 0
    for j = 0 to 5 step 2 do
      y_loc(i) := y_loc(i) + A_loc(i,j)x_loc(j)
    endfor
  endfor
% combinazione dei risultati parziali
  send(y_loc(3:5), 1)
  recv(aus(0:2), 1)
  for i=0 to 2 do
    y_loc(i) := y_loc(i) + aus(i)
  endfor
% scambio delle componenti aggiornate
  send(y_loc(0:2), 1)
  recv(y_loc(3:5), 1)
end
```

```
% calcolo dei prodotti parziali
  for i = 0 to 5 do
    y_loc(i) := 0
    for j = 1 to 5 step 2 do
      y_loc(i) := y_loc(i) + A_loc(i,j)x_loc(j)
    endfor
  endfor
% combinazione dei risultati parziali
  recv(aus(0:2), 0)
  send(y_loc(0:2), 0)
  for i=3 to 5 do
    y_loc(i) := y_loc(i) + aus(i-3)
  endfor
% scambio delle componenti aggiornate
  recv(y_loc(0:2), 0)
  send(y_loc(3:5), 0)
end
```

Procedura 3.10 - Algoritmi per il calcolo del prodotto matrice-vettore - Distribuzione ciclica per colonne della matrice A - fine

△

La distribuzione dei dati utilizzata nell'esempio 10 è un caso particolare della *Distribuzione Ciclica a Blocchi di Colonne* in cui le colonne della matrice A sono prima divise in blocchi di dimensione nb e poi, ciascun blocco, viene assegnato ad un processore, in modo che la k-esima colonna di A appartenga al processore il cui identificativo è:

$$id = mod(\lfloor k/nb \rfloor, p)$$

dove il simbolo $\lfloor x \rfloor$ indica la parte intera di x. Nell'esempio 10 $nb = 1$ e $p = 2$.

In sintesi, in un modello message passing possiamo individuare le tre fasi algoritmiche seguenti:

```
begin
    1) distribuzione dei dati
    2) esecuzione delle operazioni concorrenti
    3) combinazione dei risultati
end
```

Sia nella I che nella II strategia, gli algoritmi risolutivi appaiono, a prima vista, differenti. Sembrerebbe quindi necessario scrivere algoritmi diversi da eseguire su ciascun processore.
Il modello di programmazione che prevede la progettazione di un algoritmo per ciascun processore prende il nome di *MPMD* (***M**ultiple **P**rogram **M**ultiple **D**ata*) e può essere implementato solo su macchine di tipo MIMD in quanto richiede un flusso di istruzioni multiplo. In tale modello ogni processore esegue un proprio algoritmo indipendentemente dagli altri.

Analizzando piú attentamente gli algoritmi mostrati nella *Procedura* 3.10 notiamo che essi si differenziano, in effetti, solo per i dati su cui ogni processore opera, mentre le operazioni da eseguire sono le stesse per entrambi i processori. Tale modello di programmazione prende il nome di *SPMD* (***S**ingle **P**rogram **M**ultiple **D**ata*).

Nel modello SPMD, ogni processore esegue lo stesso algoritmo asincronamente e la sincronizzazione avviene solo quando i processori devono scambiarsi i dati. In questo caso, appare naturale domandarsi se sia possibile scrivere un solo algoritmo adatto per tutti i processori. Vediamo quindi come, introducendo opportune variabili, sia possibile scrivere un solo algoritmo che vada bene per tutti e due i processori.

Innanzitutto, osserviamo che tutti i processori devono avere due informazioni fondamentali:

- il numero di processori
- il proprio identificativo

Tale operazione, detta *"inizializzazione dell'ambiente"*, viene denotata con l'istruzione

loc.init (p, myid)

la quale restituisce in p il numero dei processori ed in *myid* l'identificativo del processore che l'ha invocata.
Nel nostro caso, se *loc.init (p, myid)* viene invocata dai due processori, restituirà ad entrambi $p = 2$; inoltre, al primo processore verrà restituito l'identificativo $myid = 0$, mentre al secondo $myid = 1$. Tali informazioni caratterizzano l'ambiente computazionale parallelo.

Vediamo, quindi come, utilizzando queste due variabili, l'algoritmo che realizza il prodotto matrice-vettore, mostrato nelle *Procedure* 3.11 (I Strategia) e 3.12 (II Strategia), può essere riscritto in un unico modo.
L'algoritmo calcola innanzitutto la dimensione locale, cioè il numero di righe (o di colonne) della matrice assegnata ai processori. Tale dimensione è il rapporto tra il numero complessivo di righe (colonne) della matrice e il numero di processori $k = n/p$ [7]. Le variabili k_1 e k_2 specificano per ciascun processore l'indice rispettivamente della prima e dell'ultima riga (colonna) del blocco di matrice da assegnare ad ogni processore[8]. Nella seconda strategia le variabili k_1 e k_2 indicano anche le righe del vettore x da assegnare ad ogni processore. Vengono quindi distribuiti i dati e calcolate le componenti del vettore y_{loc}. Ricordiamo che, mentre nella prima strategia il parallelismo viene introdotto nel ciclo esterno su i, nella seconda

[7] Per semplicità di calcolo si assume che n sia un multiplo intero di p.

[8] Se ad esempio $n = 9$ e $p = 3$ allora ad ogni processore verranno assegnate $k = n/p = 3$ righe (colonne). In particolare la prima riga (colonna) della matrice che deve essere assegnata al processore P_0 è la riga (colonna) $k1 = k * myid = 3 * 0 = 0$. La prima riga (colonna) della matrice che deve essere assegnata al processore P_1 è la riga (colonna) $k1 = k * myid = 3 * 1 = 3$. La prima riga (colonna) della matrice che deve essere assegnata al processore P_2 è la riga (colonna) $k1 = k * myid = 3 * 2 = 6$.

strategia il parallelismo viene introdotto nel ciclo interno su j, cioè nella prima si effettuano prodotti scalari in parallelo, nella seconda i prodotti vengono *"spezzati"* in somme parziali. Avviene quindi lo scambio di informazioni tra i processori: ciascuno calcola l'identificativo del processore con cui comunicare, *proccom*, e gli indici dei blocchi che deve inviare e ricevere: $l1$, $l2$, $l3$ ed $l4$. Aggiornato il vettore y_{loc} ogni processore possiede una copia del vettore soluzione y.

Nelle **Procedure** 3.11, 3.12 - 3.13 sono riportati gli algoritmi della I e II strategia rispettivamente, questa volta però osserviamo che l'algoritmo è lo stesso indipendentemente dal processore dal quale viene eseguito.

Procedure MATVET(A,x,y,n) I STRATEGIA

```
begin
% inizializzazione dell'ambiente
    loc.init(p,myid)
% numero di righe da assegnare a ciscun processore
    k := n/p
% determinazione degli indici delle righe da distribuire
    k1 := myid × k
    k2 := k1 + k − 1
% distribuzione dei dati
    A_loc := A(k1 : k2, :)
    x_loc := x(:)
    y_loc := y(0 : n − 1)
% calcolo delle componenti di y_loc relative ai dati acquisiti
   for i = 0 to k − 1 do
      y_loc(k ∗ myid + i) := 0
      for j = 0 to n − 1 do
         y_loc(k ∗ myid + i) := y_loc(k ∗ myid + i) + A_loc(i, j)x_loc(j)
      endfor
   endfor
% calcolo dell'identificativo del processore con cui comunicare
% e degli indici degli elementi da spedire e ricevere
    if(myid = 0) then
       proccom := myid + 1
       l1 := 0; l2 := n/2 − 1; l3 := n/2; l4 := n − 1
     else
       proccom = myid − 1
       l1 := n/2; l2 := n − 1; l3 := 0; l4 := n/2 − 1
    endif
% scambio delle componenti di y_loc calcolate
    send(y_loc(l1 : l2), idcom)
    recv(y_loc(l3 : l4), idcom)
end
```

Procedura 3.11 - Procedura per il prodotto matrice-vettore su $p = 2$ processori utilizzando la prima strategia.

Procedure MATVET(A,x,y,n) II STRATEGIA

begin
% inizializzazione dell'ambiente
loc.init(p,myid)
% numero di colonne da assegnare a ciscun processore
$k := n/p$
% determinazione degli indici delle colonne di A
% e delle righe di x da distribuire
$k1 := myid \times k$
$k2 := k1 + k - 1$
% distribuzione dei dati
$A_{loc} := A(:, k1 : k2)$
$x_{loc} := x(k1 : k2)$
$y_{loc} := y(0 : n-1)$
% calcolo delle somme parziali per il vettore y_{loc}
for $i = 0$ **to** $n-1$ **do**
$y_{loc}(i) := 0$
for $j = 0$ **to** $k-1$ **do**
$y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$
endfor
endfor
% calcolo dell'identificativo del processore con cui comunicare
% e degli indici degli elementi da spedire e ricevere
if$(myid = 0)$ **then**
$proccom := myid + 1$
$l1 := 0;\ l2 := n/2 - 1;\ l3 := n/2;\ l4 := n-1$
else
$proccom := myid - 1$
$l1 := n/2;\ l2 := n-1;\ l3 := 0;\ l4 := n/2 - 1$
endif
% combinazione delle somme parziali
call $SUM2(proccom, y_{loc}, l1, l2, l3, l4)$
end

Procedura 3.12 - Procedura per il prodotto matrice-vettore su $p = 2$ processori utilizzando la seconda strategia.

Procedure $SUM2(idcom, y_{loc}, l1, l2, l3, l4)$

begin

% il proc. invia ad idcom la prima (seconda) metà del
% "proprio" vettore da aggiornare e ne riceve la seconda (prima)
% metà da idcom

send$(y_{loc}(l3 : l4), idcom)$
recv$(aus(0 : n/2 - 1), idcom)$
% ciascun processore esegue la somma di n/2 componenti
for $i = l1$ **to** $l2$ **do**
    $y_{loc}(i) := y_{loc}(i) + aus(i - l1)$
endfor

% i processori si scambiano parti del vettore
% y_{loc} aggiornato

send$(y_{loc}(l1 : l2), idcom)$
recv$(y_{loc}(l3 : l4), idcom)$
end

Procedura 3.13 - Procedura per la combinazione dei risultati su $p = 2$ processori.

Generalizziamo le due strategie per il prodotto di una matrice per un vettore al caso in cui $A \in \Re^{n \times n}$ con $n = p \times k$ e $k \in \aleph$ [9].

I Strategia

Si suddivide la matrice A in p blocchi di k righe ed n colonne:

$$\begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{p-1} \end{bmatrix} \cdot x = y \; con \; A_i = \begin{bmatrix} a_{i*\frac{n}{p},0} & \dots & a_{i*\frac{n}{p},n-1} \\ \vdots & \dots & \vdots \\ a_{i*\frac{n}{p}+\frac{n}{p}-1,0} & \dots & a_{i*\frac{n}{p}+\frac{n}{p}-1,n-1} \end{bmatrix}$$

[9] Si considera questo caso per semplicità. In generale se n non è multiplo di p, ovvero $n = p \cdot k + r$, con $r < p$ si dovranno distribuire le r righe (colonne) rimanenti ai processori e tener conto del fatto che, essendo $r < p$, non tutti i processori avranno queste ultime righe (colonne).

Il prodotto

$$Ax = y$$

viene decomposto nei prodotti:

$$A_i \cdot x = y_i \;\; con\; i = 0, \ldots, p-1\;,\; A_i \in \Re^{k \times n},\; x \in \Re^n \; e \; y_i \in \Re^k$$

Riscriviamo (*Procedura* 3.14) l'algoritmo sequenziale, mettendo in evidenza le operazioni matriciali relative a ciascun blocco; si ha, in questo modo, la versione *"a blocchi"* dell'algoritmo nella *Procedura* 3.11 (I Strategia).

Procedure MATVET(A,x,y,p,n) A Blocchi di Righe

```
integer p, i, n
real A(n, n), x(n), y(n)
begin
% calcolo delle componenti di y suddivise in p blocchi
  for i = 0 to p − 1 do
    y_i = 0
    y_i = A_i · x
  endfor
end
```

Procedura 3.14 - Procedura che esegue il prodotto matrice × vettore considerando i blocchi riga in cui è stata scomposta la matrice A.

La riformulazione dell'algoritmo in termini di operazioni sui blocchi consente, in modo naturale, la formulazione dell'algoritmo parallelo ottenuto assegnando l'operazione su ciascun blocco ad un processore. Più precisamente, il processore P_i, per $0 \leq i \leq p-1$, ha:

- il blocco $A_i = \begin{bmatrix} a_{i*\frac{n}{p},0} & \dots & a_{i*\frac{n}{p},n-1} \\ \vdots & \dots & \vdots \\ a_{i*\frac{n}{p}+\frac{n}{p}-1,0} & \dots & a_{i*\frac{n}{p}+\frac{n}{p}-1,n-1} \end{bmatrix}$ di A;

- tutte le componenti del vettore x;

- tutte le componenti del vettore y;

Tutti i P_i calcolano un prodotto matrice-vettore:

$$y_i = A_i \cdot x$$

Affinché ciascun processore abbia tutte le componenti del vettore prodotto y è quindi necessario effettuare delle comunicazioni tra i processori.

Se, invece, distribuiamo la matrice A ciclicamente in blocchi di righe di dimensione 1 tra i p processori, il processore P_i, per $0 \leq i \leq p-1$, ha:

- il blocco A_i costituito dalle righe k , per $k = 0, \dots, n-1$, tali che:

$$mod(k, p) = i$$

- tutte le componenti del vettore x;

- tutte le componenti del vettore y;

Quindi, anche in questo caso, tutti i P_i calcolano un prodotto matrice-vettore:

$$y_i = A_i \cdot x$$

in cui il vettore y_i contiene le componenti k, per $k = 0, \dots, n-1$,

del vettore risultante, tali che :

$$mod(k,p) = i$$

e sono, quindi, necessarie delle comunicazioni tra i processori affinché ciascuno abbia tutte le componenti del vettore prodotto y.

II Strategia

Si suddivide la matrice A in p blocchi di n righe e k colonne ed il vettore x in p blocchi di k componenti:

$$\begin{bmatrix} A_0 & A_1 & \dots & A_{p-1} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{p-1} \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{p-1} \end{bmatrix}$$

con

$$A_i = \begin{bmatrix} a_{0,i*\frac{n}{p}} & \dots & a_{0,i*\frac{n}{p}+\frac{n}{p}-1} \\ \vdots & \dots & \vdots \\ a_{n-1,i*\frac{n}{p}} & \dots & a_{n-1,i*\frac{n}{p}+\frac{n}{p}-1} \end{bmatrix}, \; x_i = \begin{bmatrix} x_{i*\frac{n}{p}} \\ \vdots \\ x_{i*\frac{n}{p}+\frac{n}{p}-1} \end{bmatrix}$$

e

$$r_i = \begin{bmatrix} r_i^0 \\ \vdots \\ r_i^{n-1} \end{bmatrix}$$

Il prodotto

$$Ax = y$$

viene decomposto nei prodotti:

$$A_i \cdot x_i = r_i \;\; con \; A_i \in \Re^{n \times k}, x_i \in \Re^k \; e \; r_i \in \Re^n$$

e riscrivendo (*Procedura* 3.15) l'algoritmo analogamente a quanto fatto in precedenza, cioè mettendo in evidenza le operazioni matriciali relative a ciascun blocco, si ha la versione *"a blocchi"* dell'algoritmo nella *Procedura* 3.12 (II Strategia).

Procedure MATVET(A,x,y,p,n) A Blocchi di Colonne

```
integer p, i, n
real A(n,n), x(n), y(n)
begin
% inizializzazione a zero del vettore y
  y = 0
% calcolo del vettore y
  for i = 0 to p − 1 do
% calcolo del contibuto i_esimo per la determinazione di y
    r_i = A_i · x_i
% aggirnamento del vettore y
    y = y + r_i
  endfor
end
```

Procedura 3.15 - Procedura che esegue il prodotto matrice × vettore considerando i blocchi colonna in cui è stata scomposta la matrice A.

A partire dalla formulazione a blocchi dell'algoritmo, nasce in modo naturale l'algoritmo parallelo corrispondente ottenuto assegnando l'operazione su ciascun blocco ad un processore. Più precisamente, il processore P_i, $0 \leq i \leq p-1$, ha:

- il blocco $A_i = \begin{bmatrix} a_{0,i*\frac{n}{p}} & \dots & a_{0,i*\frac{n}{p}+\frac{n}{p}-1} \\ \vdots & \dots & \vdots \\ a_{n-1,i*\frac{n}{p}} & \dots & a_{n-1,i*\frac{n}{p}+\frac{n}{p}-1} \end{bmatrix}$ di A;

- il blocco $x_i = \begin{bmatrix} x_{i*\frac{n}{p}} \\ \vdots \\ x_{i*\frac{n}{p}+\frac{n}{p}-1} \end{bmatrix}$ di x;
- tutte le componenti del vettore y.

Tutti i processori P_i calcolano:

$$r_i = A_i \cdot x_i \ , \quad con \ \ A_i \in \Re^{n \times k} \ , \ x_i \in \Re^k \ e \ r_i \in \Re^n$$

Quindi è necessario, per ottenere il risultato finale, effettuare la somma delle componenti dei p risultati locali distribuiti tra i p processori:

$$y_j = \sum_{i=0}^{p-1} r_i^j \ \ con \ j = 0, \ldots, n-1$$

Tale operazione richiede delle comunicazioni tra i processori.

Infine, se distribuiamo la matrice A ciclicamente in blocchi di colonne di dimensione 1 tra i p processori, il processore P_i, per $0 \leq i \leq p-1$, ha:

- il blocco A_i costituito dalle colonne k , per $k = 0, \ldots, n-1$, tali che:

$$mod(k, p) = i$$

- le componenti k , per $k = 0, \ldots, n-1$, del vettore x tali che:

$$mod(k, p) = i$$

- tutte le componenti del vettore y;

Anche in questo caso, tutti i processori P_i calcolano un prodotto

matrice-vettore:

$$r_i = A_i \cdot x_i$$

in cui il vettore r_i contiene le componenti parziali k , per $k = 0, \ldots, n-1$, del vettore risultante y, tali che :

$$mod(k, p) = i$$

Quindi è necessario, per ottenere il vettore finale y, effettuare la somma delle componenti dei p risultati locali r_i distribuiti tra i p processori:

$$y_j = \sum_{i=0}^{p-1} r_i^j \;\; con\ j = 0, \ldots, n-1$$

3.2 Somma di $p = 2^m$ vettori

Nel caso di due processori abbiamo visto che la somma di due vettori è descritta dall'algoritmo riportato nella *Procedura* 3.13 .

Generalizzando l'algoritmo al caso in cui i processori sono $p = 2^m$, saranno necessari $m = log_2 p$ passi. Ad ogni passo, coppie distinte di processori comunicano. Vediamo come determinare ad ogni passo gli identificativi delle coppie di processori che devono comunicare tra di loro. Aiutiamoci con un esempio.

♣ Esempio 11

Determinazione ad ogni passo delle coppie di identificativi di processori che comunicano tra loro.

Consideriamo il caso in cui si hanno $p = 4 = 2^2$ processori (Fig. 3.13).

I passo

$P_0 = 00$ comunica con $P_1 = 01$

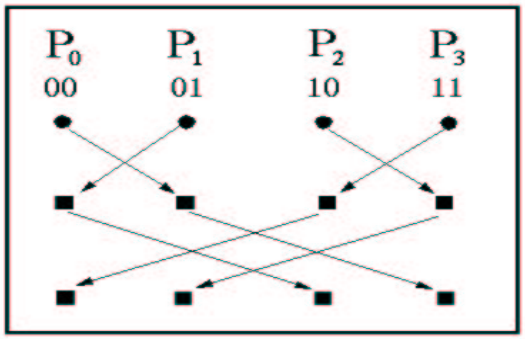


Figura 3.13: *Nella figura sono evidenziate le comunicazioni ad ogni passo tra i processori in base al loro identificativo, nel caso in cui* $p = 4$.

$P_2 = 10$ comunica con $P_3 = 11$

comunicano i processori il cui identificativo differisce solo nel *primo* bit.

II passo

$P_0 = 00$ comunica con $P_2 = 10$

$P_1 = 01$ comunica con $P_3 = 11$

comunicano i processori il cui identificativo differisce solo nel *secondo* bit.

Consideriamo ora il caso in cui $p = 8 = 2^3$.

I passo

$P_0 = 000$ comunica con $P_1 = 001$

$P_2 = 010$ comunica con $P_3 = 011$

$P_4 = 100$ comunica con $P_5 = 101$

$P_6 = 110$ comunica con $P_7 = 111$

comunicano i processori il cui identificativo differisce solo nel *primo* bit.

II passo

$P_0 = 000$ comunica con $P_2 = 010$

$P_1 = 001$ comunica con $P_3 = 011$

$P_4 = 100$ comunica con $P_6 = 110$

$P_5 = 101$ comunica con $P_7 = 111$

comunicano i processori il cui identificativo differisce solo nel *secondo* bit.

III passo

$P_0 = 000$ comunica con $P_4 = 100$

$P_1 = 001$ comunica con $P_5 = 101$

$P_2 = 010$ comunica con $P_6 = 110$

$P_3 = 011$ comunica con $P_7 = 111$

comunicano i processori il cui identificativo differisce solo nel *terzo* bit.

△

In generale, al passo i-esimo comunicano i processori il cui identificativo differisce solo nel bit $(m - i + 1)$-mo.

In definitiva, l'algoritmo per il prodotto matrice-vettore su un calcolatore MIMD a memoria distribuita con $p = 2^m$ processori ed $n = kp$ è mostrato nelle *Procedure* 3.16, 3.17, 3.18, 3.19 . In particolare, in Fig. 3.14 è mostrato inizialmente uno schema grafico delle macro operazioni (flow chart).

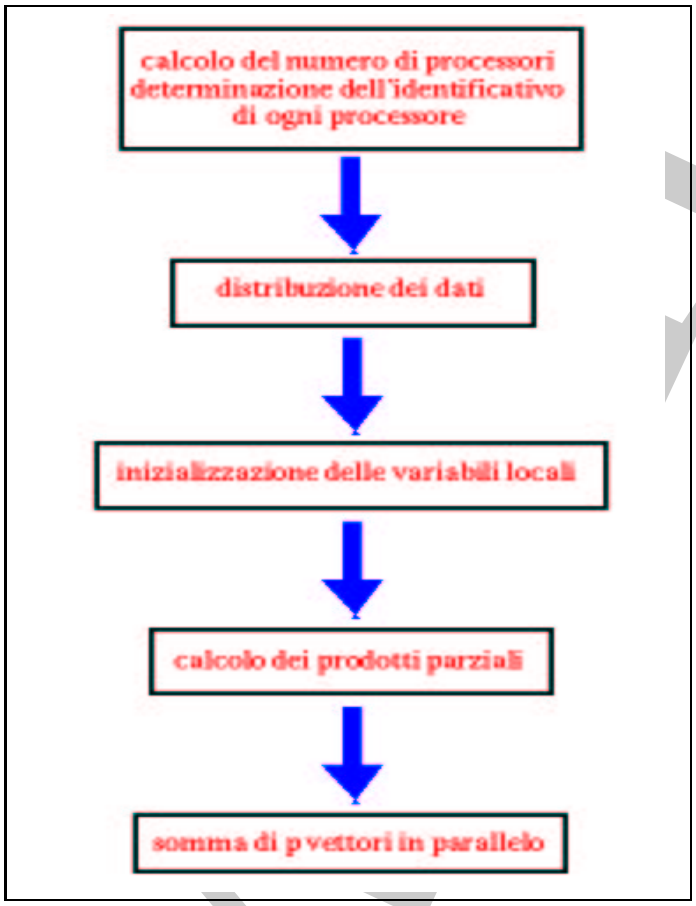


Figura 3.14: *Schema generale delle fasi fondamentali per il calcolo del prodotto matrice per vettore.*

Algoritmo MATVET , I STRATEGIA $p = 2^m, n = kp$

```
begin procedure matvet(n,p,A,x,y)
  % calcolo del numero di processori p
  % e del proprio identificativo myid
  loc.init(p, myid)
  % k=numero di righe distribuite a ciascun processore
  k := n/p
  % k1=indice della prima riga distribuita
  % al processore di id. myid
  k1 := k · myid
  % k2=indice dell'ultima riga distribuita
  % al processore di id. myid
  k2 := k1 + k − 1
  % distribuzione di dati
  % inizializzazione delle variabili locali
  A_loc := A(k1 : k2, :)
  x_loc := x(0 : n − 1)
  y_loc := y(0 : n − 1)
  % calcolo dei prodotti parziali
  for i=0 to k-1 do
      y_loc(i) := 0
      for j=0 to n-1 do
          y_loc(i) := y_loc(i) + A_loc(i, j)x_loc(j)
      endfor
  endfor
  % chiamata alla procedura che effettua il
  % collezionamento dei dati
  call collezione (n,p,myid,y)
end
```

Procedura 3.16 - Procedura per il prodotto matrice × vettore su p processori utilizzando la prima strategia (distribuzione della matrice per blocchi di righe). Viene prima determinato il numero di righe k da distribuire a ciascun processore. Vengono quindi determinati gli indici della prima e dell'ultima riga da distribuire a ciascun processore e vengono distribuiti i dati. Ciascun processore calcola le proprie componenti del vettore y e viene chiamata la procedura ***collezione*** *che colleziona i dati distribuiti sui processori.*

Algoritmo per collezionare i dati distribuiti su p processori

begin collezione (n,p,myid,y)
for i=1 to $log_2 p$ do
% *idcom*=**identificativo del processore con cui comunicare**
% *resto*=**i-mo bit di** *myid*
% *numel*= **numero di elementi che devono essere scambiati.**
$resto := myid - \lfloor \frac{myid}{2^i} \rfloor \times 2^i$
numel=$2^i * n/p$
% $\mathbf{P}_{myid}$ **comunica con** $\mathbf{P}_{myid+2^{i-1}}$
if ($resto < 2^{i-1}$) then
$idcom := myid + 2^{i-1}$
l1=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p}$, l2=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p} + numel - 1$
l3=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p} + numel$, l4=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p} + 2 * numel - 1$
% **oppure con** $\mathbf{P}_{myid-2^{i-1}}$
else
$idcom := myid - 2^{i-1}$
l1=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p}$, l2=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p} + numel - 1$
l3=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p} - numel$, l4=$\lfloor \frac{myid}{2^{i-1}} \rfloor * 2^{i-1} * \frac{n}{p} - 1$
endif
% **scambio dei dati**
send(y(l1:l2,idcom)
recv(y(l3:l4),idcom)
endfor
end

Procedura 3.17 - Procedura per il collezionamento dei dati distribuiti su p processori. Ad ogni passo viene determinato l'identificativo del processore con cui comunicare e il numero di elementi che devono essere scambiati. Vengono inoltre determinati gli indici l1 ed l2 della prima e dell'ultima componente da inviare e gli indici l3 ed l4 della prima e dell'ultima componente che devono essere ricevute. Infine vengono effettuate le comunicazioni.

Algoritmo MATVET , II STRATEGIA $p = 2^m, n = kp$

```
begin procedure matvet(n,p,A,x,y)
 % calcolo del numero di processori p
 % e del proprio identificativo myid
 % k=numero di colonne distribuite a ciascun processore
 k := n/p
 % k1=indice della prima colonna distribuita
 % al processore di id. myid
 k1 := k · myid
 % k2=indice dell'ultima colonna distribuita
 % al processore di id. myid
 k2 := k1 + k − 1
 % distribuzione di dati
 % inizializzazione delle variabili locali
 A_loc := A(:, k1 : k2)
 x_loc := x(k1 : k2)
 y_loc := y(0 : n − 1)
 % calcolo dei prodotti parziali
 for i=0 to n-1 do
     y_loc(i) := 0
     for j=0 to k-1 do
         y_loc(i) := y_loc(i) + A_loc(i, j)x_loc(j)
     endfor
 endfor
 % chiamata alla procedura che effettua la
 % somma di p vettori in parallelo
 call somma (n,p,myid,y)
end
```

Procedura 3.18 - Procedura per il prodotto matrice × vettore su p processori utilizzando la seconda strategia (distribuzione della matrice per blocchi di colonne). Viene prima determinato in numero di colonne k da distribuire a ciascun processore. Vengono quindi determinati gli indici della prima e dell'ultima colonna da distribuire a ciascun processore e vengono distribuiti i dati. Ciascun processore calcola i prodotti parziali delle componenti del vettore y e viene chiamata la procedura somma che colleziona i dati distribuiti sui processori.

Algoritmo somma di p vettori

```
begin somma (n,p,myid,y)
  for i=1 to log2p do
  % idcom=identificativo del processore con cui
  % effettuare la somma
  % resto=i-mo bit di myid
   resto := myid - ⌊myid/2^i⌋ × 2^i
  % P_myid comunica con P_myid+2^(i-1)
   if (resto < 2^(i-1)) then
      idcom := myid + 2^(i-1)
      l1 := 0; l2 := n/2 - 1; l3 := n/2; l4 := n - 1
  % oppure con P_myid-2^(i-1)
    else
      idcom := myid - 2^(i-1)
      l1 := n/2; l2 := n - 1; l3 := 0; l4 := n/2 - 1
   endif
  % procedura per la somma di 2 vettori
  % distribuiti tra 2 processori
   procedure SUM2(idcom,y,l1,l2,l3,l4)
   endfor
end
```

Procedura 3.19 - Procedura per la somma di p vettori distribuiti su p processori. Ad ogni passo viene determinato l'identificativo del processore con cui comunicare. Vengono inoltre determinati gli indici l1 ed l2 che individuano la parte del vettore da inviare e gli indici l3 ed l4 che individuano la parte del vettore che deve essere ricevuto. Infine vengono effettuate le comunicazioni.

Analisi delle prestazioni dell'algoritmo

Sia t_{calc} il tempo impiegato per l'esecuzione di un'operazione floating point e t_{com} il tempo necessario alla comunicazione di un dato floating point. Si osserva che nella II strategia, per il calcolo dei prodotti parziali ogni processore effettua il calcolo di y_{loc} nel modo

seguente:

$$y_{loc}(i) := y_{loc}(i) + A_{loc}(i,j)x_{loc}(j)$$

per $i = 0, \ldots, n-1$ e $j = 0, \ldots, k-1$, che richiede

$$T_{calc} = 2 \cdot n \cdot k \cdot t_{calc}$$

avendo indicato con T_{calc} il tempo totale di calcolo; n è l'ordine della matrice A e la dimensione di x ed y, mentre k è il numero di righe o colonne distribuite a ciascun processore.
Tenuto conto che $k = n/p$, si ha:

$$T_{calc} = 2 \cdot \frac{n^2}{p} t_{calc}$$

Inoltre, ad ogni passo i processori scambiano tra loro $n/2$ componenti di y_{loc}. Tenuto conto che il numero di passi è $log_2 p = m$, vengono effettuate

$$T_{com} = m \cdot \frac{n}{2} \ comunicazioni$$

avendo indicato con T_{com} il tempo totale di comunicazione.
In definitiva si ha:

$$T_{calc} = 2 \cdot \frac{n^2}{p} t_{calc}$$

$$T_{com} = m \cdot \frac{n}{2} t_{com}$$

3.3 Lo standard MPI

Il criterio per progettare algoritmi paralleli su calcolatori MIMD è quello di pensare ad un insieme di processi concorrenti che coordinano la proprià attività attraverso la comunicazione di messaggi. Tale modello di programmazione è noto con il nome di MESSAGE PASSING.

PVM e MPI sono due delle piú note librerie di software alla base del paradigma del message passing: sono cioè un insieme di funzioni che consentono ai processori di scambiarsi informazioni[10].

PVM ([25]) è l'acronimo di Parallel Virtual Machine e consente di vedere una rete di calcolatori UNIX come un singolo calcolatore parallelo a memoria distribuita (la macchina virtuale). Lo sviluppo di PVM è iniziato nel 1989 all'Oak Ridge National Laboratory (ORNL).
MPI è l'acronimo di *"Message Passing Interface"* [18]. MPI nasce dalla necessità di definire uno standard unico per le librerie di message passing per gli Stati Uniti e l'Europa. In Europa dal 1991 il *Gruppo di Interesse Speciale sulla Standardizzazione dei Linguaggi per Macchine MIMD a Memoria Distribuita dell'ESPRIT* si era preoccupato di disegnare uno standard per il message passing. Questo intento portó al progetto SHAPES, che tentava di unificare i modelli di programmazione di UBIK ASI e di PARMACS[a]. Purtroppo i due approcci erano troppo differenti per essere unificati. PARMACS fu una delle basi per MPI.
Negli Stati Uniti già dai primi anni '90 erano presenti sul mercato vari calcolatori a memoria distribuita e alcuni gruppi di ricerca avevano sviluppato sistemi di message passing che cercavano di fornire portabilità attraverso piattaforme differenti. PVM era uno di questi.
Jack Dongarra e Tony Hey furono tra i primi a riconoscere la necessità di coordinare i tentativi di standardizzazione europei e di oltre oceano al fine di evitare la scelta di standard differenti. Nell'estate del 1992 Jack Dongarra, Rolf Hempel, Tony Hey e David Walker decisero di definire uno standard per il message passing.

[a] PARMACS sta per PARallel MACroS

[10] Gli indirizzi relativi alle librerie PVM ed MPI sono risperttivamente:
http : *//www.csm.ornl.gov/pvm/pvm_home.html*
http : *//www − unix.mcs.anl.gov/mpi*

Questo processo ebbe inizio con il *Workshop su Standard per il Message Passing in Ambiente a Memoria Distribuita* che ebbe luogo il 29-30 Aprile del 1992 a Williamsburg, Virginia (USA). Nel Workshop, sponsorizzato dal CRPC (Center for Research on Parallel Computing) della Rice University, furono discusse differenti interfacce per il message passing ed esaminate le principali caratteristiche che avrebbe dovuto avere uno standard per il message passing. Su proposta di Ken Kennedy della Rice University fu istituito un gruppo di lavoro per lo sviluppo dello standard.
Nell'Ottobre del 1992 fu presentata una prima versione dello standard: MPI-0. MPI-0 comprendeva le principali caratteristiche individuate durante il Workshop, ma mancavano le routine per le comunicazioni collettive. Fu deciso allora di dare al processo di standardizzazione un'impronta piú formale. Fu quindi istituito l'MPI Forum e furono create sottocommissioni per i settori piú importanti dello standard.
Gli obiettivi originali dell'MPI Forum erano di sviluppare uno standard per il message passing che fosse largamente utilizzato, pratico, portabile, efficiente ed estensibile. Più precisamente disegnare un'interfaccia che:

- potesse essere utilizzata nei progetti di sviluppo software a vari livelli, dalle applicazioni per gli utenti alle librerie parallele ottimizzate;
- fosse non troppo differente da quelle già esistenti come PARMACS o PVM;
- potesse essere implementata su differenti piattaforme, senza sostanziali modifiche nelle comunicazioni e nel software di sistema;
- consentisse l'implementazione su sistemi eterogenei;
- realizzasse chiamate al C e al Fortran rendendo contemporaneamente la semantica dell'interfaccia indipendente dal linguaggio;

Per concludere questo ambizioso progetto in poco tempo, l'MPI Forum si è incontrato per i primi nove mesi del 1993 ogni sei settimane a Dallas nel Texas. Alla conferenza Supercomputing del 1993 a Portaland, nell'Oregon, fu presentato un draft della nuova versione MPI-1. È seguito un periodo di consultazione, fino al Febbraio del 1994, durante il quale le persone erano invitate a commentare le specifiche. A Gennaio del 1994 fu organizzata una riunione dell'MPI Forum all'INRIA Sophia Antiplos in Francia, per consentire anche alla comunità dell'High Performance Computing Europea di contribuire al nuovo standard. Infine, il 5 Maggio 1994, la versione 1.0 di MPI fu distribuita su Internet. Negli anni successivi sono state rilasciate nuove versioni di MPI.

Lo standard MPI comprende le applicazioni seguenti:

- comunicazioni punto-punto;
- operazioni collettive;
- gruppi di processori;
- domini di comunicazione;
- topologie di processori;
- gestione dell'ambiente;
- interfaccia per il profiling;
- chiamate al Fortran e al C.

Lo scambio di informazioni tra due o piú processori avviene attraverso la comunicazione di messaggi:

- comunicazioni *"uno a uno"*;
- comunicazioni *"collettive"*.

Le comunicazioni *"uno a uno"* coinvolgono solo due processori. Le comunicazioni *"collettive"*, invece, coinvolgono piú processori contemporaneamente. Sono possibili tipi differenti di comunicazioni collettive:

- comunicazione di uno a piú di uno (broadcast);
- scambio di informazioni tra piú processori (operazioni di riduzione, come la ricerca del massimo di una lista distribuita, somma degli elementi di un vettore distribuito, ecc.).

MPI fornisce le routine che implementano nella maniera piú efficiente le varie modalità di comunicazioni esaminate.

Nel disegno di un algoritmo basato su MPI la prima cosa da fare è richiamare il file contenente le definizioni necessarie al preprocessore per l'utilizzo di MPI. Questa operazione si realizza mediante la direttiva (MACRO):

```
♯ include "/usr/mpich/include/mpi.h"
```

Le altre operazioni di base sono le seguenti:

1. *Inizializzazione di MPI*

   ```
   MPI_Init()
   ```

   Questa routine deve essere chiamata prima di ogni altra routine MPI. Definisce l'insieme dei processori attivati (contesto).

2. *Identificativo dei processori*

   ```
   MPI_Comm_rank(MPI_Comm comm,int *menum)
   ```

   Assegna ad ogni processore del `communicator` comm l'identificativo menum. In MPI si distingue tra `contesto` e `communicator`. Il primo è un identificativo associato ad un gruppo di processori. Il secondo è un ulteriore identificativo, associato ad un contesto e racchiude tutte le informazioni dell'ambiente di comunicazione, come la topologia, quali e quanti processori sono coinvolti, etc.

   MPI_COMMON_WORLD indica il `communicator` a cui appartengono tutti i processori attivati e non può essere alterato dopo l'inizializzazione.

3. *Numero di processori del communicator*

   ```
   MPI_Comm_size(MPI_COMM comm,int *nproc)
   ```

Restituisce ad ogni processore del communicator comm il numero totale di processori del contesto. Permette di conoscere quati processori stanno concorrendo per una determinata operazione.

4. *Termine di un programma MPI*

```
MPI_Finalize()
```

Determina la fine del programma MPI. Dopo questa routine non è possibile richiamare nessuna altra routine MPI.

5. *Spedizione di messaggi*

```
MPI_Send(void *buf,int count,
         MPI_Datatype datatype,int dest,
         int tag,MPI_Comm comm)
```

Spedisce i primi count elementi di buf, del tipo datatype, al processore dest. L'identificativo tag individua univocamente il messaggio nel contesto comm.

6. *Ricezione di un messaggio*

```
MPI_Recv(void *buf,int count,
         MPI_Datatype datatype,int source,
         int tag,MPI_Comm comm,
         MPI_Status *status)
```

Riceve i primi count elementi di buf, del tipo datatype, dal processore source. L'identificativo tag individua univocamente il messaggio nel contesto comm. status è l'indicatore di errore.

A scopo illustrativo riportiamo di seguito un programma scritto in linguaggio C che utilizza le routine di MPI per eseguire il prodotto matrice vettore secondo la I Strategia.

```
#include <stdio.h>
#include "mpi.h"
main(int argc,char *argv){
  int menum, nproc;
  int n,nloc,tag,i,j,k,rest;
  int passi,proc,step,tmp;
  float *x,**a,**aloc,*y,*yloc,sum;
  MPI_Status status;

  /* Inizializzazione dell'ambiente MPI */
  MPI_Init(&argc,&argv);
  /* Calcolo del numero di processori */
  MPI_Comm_size(MPI_COMM_WORLD,&nproc);
  /* Calcolo dell'identificativo */
  MPI_Comm_rank(MPI_COMM_WORLD,&menum);
  tag=1;
   /* Il processore P0 legge la matrice a */
  if (menum == 0 ){
      printf("Inserire n \n");
      scanf("%d",&n);
      y = (float*)calloc(n,sizeof(float));
      a = (float**)calloc(n,sizeof(float *));
      for (i=0; i<n; i++)
           *(a+i) = (float*)calloc(n,sizeof(float));
      printf("Inserire a \n");
      for (i=0; i<n; i++)
              for (j=0; j<n; j++)
                    scanf("%f",*(a+i)+j);
  }
  /* Il processore P0 esegue un Broadcast del dato n */
  MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);

   /* Calcolo della dimensione locale */
  nloc = n / nproc;
   /* Calcolo del resto */
  rest = n %nproc;
   /* I primi rest processori si distribuiscono rest */
  if (rest != 0 && menum < rest)
     nloc++;
  aloc = (float**)calloc(nloc,sizeof(float *));
  for (i=0; i<nloc; i++)
       *(aloc+i) = (float*)calloc(n,sizeof(float));
  x = (float*)calloc(n,sizeof(float));
  yloc = (float*)calloc(nloc,sizeof(float));
   /* Il processore P0 legge il vettore x */
  if (menum == 0){
       printf("Inserire x \n");
       for (i=0; i<n; i++)

```

Procedura 3.20 - Procedura che esegue il prodotto matrice per vettore secondo la I strategia - continua

```
50
51          scanf("%f",x+i);
52      }
53      /* Il processore P0 esegue un Broadcast del vettore x */
54      MPI_Bcast(x,n,MPI_FLOAT,0,MPI_COMM_WORLD);
55
56      /* Distribuzione da parte di 0 dei blocchi di righe della matrice a */
57      if (menum == 0){
58         aloc = a;
59         step =nloc;
60         for (i=1; i<nproc; i++){
61              if ( rest != 0 )
62                  if (i < rest)
63                       tmp = nloc;
64                  else
65                       tmp = nloc -1;
66               else
67                    tmp=nloc;
68               for (j=0; j<tmp; j++ ){
69                    tag=22+i+j;
70                    MPI_Send(*(a+step+j),n,MPI_FLOAT,i,tag,MPI_COMM_WORLD); }
71               step += tmp;
72           }
73       }
74       else{
75      /* Ricezione dei blocchi di righe della matrice a */
76                for (i=0; i<nloc; i++ ){
77                     tag = 22+menum+i;
78                     MPI_Recv(*(aloc+i),n,MPI_FLOAT,0,tag,MPI_COMM_WORLD,&status);
79                }
80       }
81
82       /* Calcolo dei prodotti locali */
83      for (i=0; i<nloc; i++){
84           *(yloc+i) = 0;
85           for (j=0; j<n; j++){
86                *(yloc+i) += *(*aloc+i)+j * *(x+j);
87           }
88      }
89
90       /* Il processore P0 colleziona opportunamente i dati  che riceve in y */
91      if (menum == 0){
92         for (k=0; k<nloc; k++){
93              *(y+k)=*(yloc+k);
94         }
95         step =nloc;
96         for (i=1; i<nproc; i++){
97              if ( rest != 0 )
98
```

Procedura 3.20 - Procedura che esegue il prodotto matrice per vettore secondo la I strategia - continua

```
99
100             if (i < rest)
101                 tmp = nloc;
102             else
103                 tmp = nloc -1;
104         else
105              tmp=nloc;
106         tag=i;
107         MPI_Recv(y+step,tmp,MPI_FLOAT,i,tag,MPI_COMM_WORLD,&status);
108         step += tmp;
109      }
110  }
111  else{
112   /* tutti i processori spediscono a 0 il loro prodotto */
113           tag=menum;
114           MPI_Send(yloc,nloc,MPI_FLOAT,0,tag,MPI_COMM_WORLD); }
115
116   /* Il processore P0 stampa il risultato finale */
117  if (menum == 0)
118     for (i=0; i<n; i++){
119        printf("prodotto %f\n",*(y+i) );}
120
121   /* Chiusura dell'ambiente MPI */
122  MPI_Finalize();
123  return 0;
124 }
```

Procedura 3.20 - Procedura che esegue il prodotto matrice per vettore secondo la I strategia - fine

3.4 Il prodotto di due matrici

Problema

Calcolo del prodotto

$$C = A \cdot B$$

con $A \in \Re^{m \times h}$, $B \in \Re^{h \times n}$ *e* $C \in \Re^{m \times n}$, *su un calcolatore tipo MIMD a memoria distribuita con p processori.*

Un algoritmo sequenziale è, ad esempio, quello che calcola la matrice prodotto componente per componente, effettuando i pro-

dotti scalari di ciascuna riga di A per ciascuna colonna di B, una componente per volta secondo un ordine prestabilito[11]:

prodotto matrice-matrice

```
begin procedure matmat(a,b,c,m,h,n)
    integer n, m, h, i, j, k
    real a(m,h), b(h,n), c(n,m)
    for i = 0 to m − 1 do
      for j = 0 to n − 1 do
          c_ij := 0
          for k = 0 to h − 1 do
              c_ij := c_ij + a_ik · b_kj
          endfor
      endfor
    endfor
end
```

Procedura 3.21 - Algoritmo sequenziale per il calcolo del prodotto matrice per matrice.

Gli elementi di C sono ottenuti mediante prodotto scalare delle righe di A e le colonne di B.

Le strategie per sviluppare un algoritmo parallelo per il prodotto di due matrici sono molteplici. Per descriverle meglio consideriamo il caso in cui $m = h = n = 6$, ovvero le 3 matrici sono quadrate di dimensione 6.

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} & c_{04} & c_{05} \\ c_{10} & c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{20} & c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{30} & c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{40} & c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\ c_{50} & c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{bmatrix} =$$

[11] È noto che l'efficienza del software che implementa l'algoritmo dipende fortemente dall'ordine degli indici, ijk. La scelta ottimale dipende dall'ambiente di calcolo (linguaggio, organizzazione della memoria, etc...). In questo momento si è scelto di prendere in esame la versione piú naturale ijk, senza badare all'efficienza computazionale.

$$= \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \cdot \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{50} & b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{bmatrix}$$

I Strategia

Poiché i prodotti scalari possono essere effettuati in modo indipendente l'uno dall'altro, è naturale introdurre una prima strategia di parallelismo nel calcolo di tali prodotti. In altre parole, poiché le componenti di C sono calcolate effettuando il prodotto scalare tra le righe di A per le colonne di B, si può pensare di suddividere A in blocchi di righe e B in blocchi di colonne. Consideriamo ad esempio la decomposizione delle matrici A e B seguenti.

$$\begin{bmatrix} A_0 \\ A_1 \end{bmatrix} \cdot \begin{bmatrix} B_0 & B_1 \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

con A_0 e $A_1 \in \Re^{\frac{m}{2} \times h}$, B_0 e $B_1 \in \Re^{h \times \frac{n}{2}}$, e le matrici $C_{ij} \in \Re^{\frac{m}{2} \times \frac{n}{2}}$, per $i, j = 0, 1$.

Ad ogni processore viene assegnato un blocco di A e un blocco di B.

processore P_0
A_0, B_0

processore P_1
A_1, B_1

Nell'esempio considerato il processore P_0 possiede gli elementi di A_0:

$$A_0 \equiv (a_{i,j}) \quad i = 0, \ldots, 2;\ j = 0, \ldots, 5$$

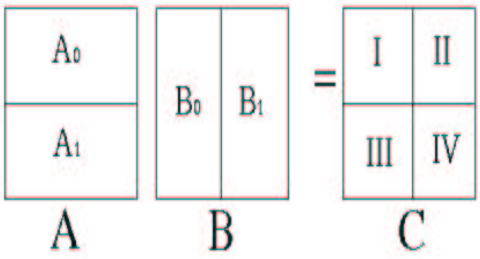


Figura 3.15: *I strategia: la matrice A viene suddivisa in due blocchi di righe, mentre la matrice B viene suddivisa in due blocchi di colonne. Con questo tipo di suddivisione dei dati ciascun processore potrà calcolare due blocchi della matrice C.*

e quelli di B_0:

$$B_0 \equiv (b_{i,j}) \quad i = 0, \ldots, 5;\ j = 0, \ldots, 2$$

Il processore P_1 possiede gli elementi di A_1:

$$A_1 \equiv (a_{i,j}) \quad i = 3, \ldots, 5;\ j = 0, \ldots, 5$$

e quelli di B_1:

$$B_1 \equiv (b_{i,j}) \quad i = 0, \ldots, 5;\ j = 3, \ldots, 5$$

Con i dati cosí distribuiti ciascun processore può calcolare una parte degli elementi di C. In particolare:

- P_0 calcola $C_{0,0} = [c_{ij}]_{i,j=0,1,2}$ dove $c_{ij} = \sum_{k=0,\ldots,5} a_{ik}b_{kj}$ per $i, j = 0, 1, 2$ cioè il blocco I di C;

- P_1 calcola $C_{1,1} = [c_{ij}]_{i,j=3,4,5}$ dove $c_{ij} = \sum_{k=0,\ldots,5} a_{ik}b_{kj}$ per $i, j = 3, 4, 5$ cioè il blocco IV di C.

Algoritmo processore P_0

```
⋮
for i=0 to 2 do
    for j=0 to 2 do
        for k = 0 to 5 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i=3 to 5 do
    for j=3 to 5 do
        for k = 0 to 5 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Procedura 3.22 - Algoritmi per il calcolo delle matrici C_{00} e C_{11}

A questo punto, se i due processori si scambiano i blocchi di B, possono calcolare i blocchi restanti di C, cioè P_0 calcolerà C_{01} e P_1 calcolerà C_{10} [12]. In memoria, quindi ciascun processore ha i blocchi di A e di B seguenti:

processore P_0	**processore P_1**
$A_0,\ B_1$	$A_1,\ B_0$

Cioè, se il processore P_0 possiede gli elementi di B_1:

$$B_1 \equiv (b_{i,j}) \quad i = 0, \ldots, 5;\ j = 3, \ldots, 5$$

e il processore P_1 possiede gli elementi di B_0:

$$B_0 \equiv (b_{i,j}) \quad i = 0, \ldots, 5;\ j = 0, \ldots, 2$$

possono entrambi effettuare i calcoli seguenti:

- P_0 calcola $C_{0,1} = [c_{ij}]_{i=0,1,2\, j=3,4,5}$ dove $c_{ij} = \sum_{k=0,\ldots,5} a_{ik} b_{kj}$ per $i = 0, 1, 2$ e $j = 3, 4, 5$ cioè il blocco II di C;
- P_1 calcola $C_{1,0} = [c_{ij}]_{i=3,4,5\, j=0,1,2}$ dove $c_{ij} = \sum_{k=0,\ldots,5} a_{ik} b_{kj}$ per $i = 3, 4, 5$ e $j = 0, 1, 2$ cioè il blocco III di C.

[12] In alternativa i processori possono anche scambiarsi i blocchi di A, in questo modo il processore P_0 calcola C_{10} ed il processore P_1 calcola C_{01}.

Algoritmo processore P_0

```
⋮
for i=0 to 2 do
    for j=3 to 5 do
        for k = 0 to 5 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i=3 to 5 do
    for j=0 to 2 do
        for k = 0 to 5 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Procedura 3.23 - Algoritmi per il calcolo delle matrici C_{10} e C_{01}

Da notare che in questa prima strategia vengono *"spezzati"* entrambi i cicli sugli indici i e j.

II Strategia

Si può anche pensare di suddividere A in blocchi di colonne e B in blocchi di righe. Avremo quindi la decomposizione delle matrici seguente:

$$\begin{bmatrix} A_0 & A_1 \end{bmatrix} \cdot \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} = C$$

con A_0 e $A_1 \in \Re^{m \times \frac{h}{2}}$, B_0 e $B_1 \in \Re^{\frac{h}{2} \times n}$, e $C \in \Re^{m \times n}$.
Ad ogni processore viene assegnato un blocco di A e un blocco di B. In particolare:

processore P_0	**processore P_1**
A_0, B_0	A_1, B_1

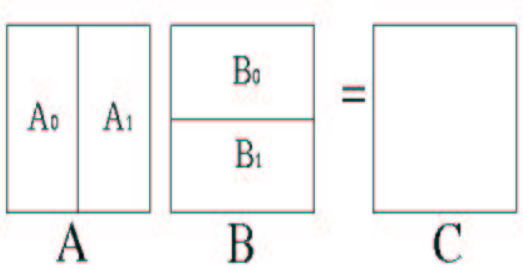


Figura 3.16: *II strategia: la matrice A viene suddivisa in due blocchi di colonne, mentre la matrice B viene suddivisa in due blocchi di righe. Con questo tipo di suddivisione dei dati ciascun processore potrà calcolare una somma parziale di tutti gli elementi della matrice C.*

Nell'esempio considerato il processore P_0 possiede gli elementi di A_0:

$$A_0 \equiv (a_{i,j}) \quad i = 0, \ldots, 5;\ j = 0, \ldots, 2$$

e quelli di B_0:

$$B_0 \equiv (b_{i,j}) \quad i = 0, \ldots, 2;\ j = 0, \ldots, 5$$

Il processore P_1 possiede gli elementi di A_1:

$$A_1 \equiv (a_{i,j}) \quad i = 0, \ldots, 5;\ j = 3, \ldots, 5$$

e quelli di B_1:

$$B_1 \equiv (b_{i,j}) \quad i = 3, \ldots, 5\ j = 0, \ldots, 5$$

Con i dati distribuiti cosí ogni processore calcola gli elementi seguenti:

- P_0 calcola $x_{ij} = \sum_{k=0,1,2} a_{ik} b_{kj} \quad i, j = 0, \ldots, 5$
- P_1 calcola $y_{ij} = \sum_{k=3,4,5} a_{ik} b_{kj} \quad i, j = 0, \ldots, 5$

dove $c_{ij} = x_{ij} + y_{ij}$.

Algoritmo processore P_0

```
⋮
for i = 0 to 5 do
    for j = 0 to 5 do
        for k = 0 to 2 do
            x_ij := x_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i = 0 to 5 do
    for j = 0 to 5 do
        for k = 3 to 5 do
            y_ij := y_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Procedura 3.24 - Algoritmi per il calcolo delle componenti parziali della matrice C.

In questa strategia abbiamo introdotto il parallelismo nel calcolo dei prodotti scalari che forniscono gli elementi della matrice C. Si può infatti notare che è stato decomposto il ciclo sull'indice k. Ogni processore contiene solo tre elementi di ogni riga della matrice A e tre elementi di ogni colonna della matrice B e non può quindi calcolare il prodotto righe-colonne completo. I processori devono allora scambiare tra loro i risultati parziali. Al secondo passo i due processori scambiano le somme parziali e le combinano:

- P_0 invia a P_1 x_{ij} e P_1 invia a P_0 y_{ij} con $i, j = 0, \ldots, 5$;
- P_0 e P_1 calcolano $c_{ij} = x_{ij} + y_{ij} \quad con \quad i, j = 0, \ldots, 5$.

III Strategia

Si possono suddividere le matrici A, B e C in blocchi di colonne e assegnare al processore P_i, con $i = 0, \ldots, p-1$, i blocchi A_i, B_i e C_i rispettivamente di dimensione $m \times \frac{h}{p}$, $h \times \frac{n}{p}$ e $m \times \frac{n}{p}$.

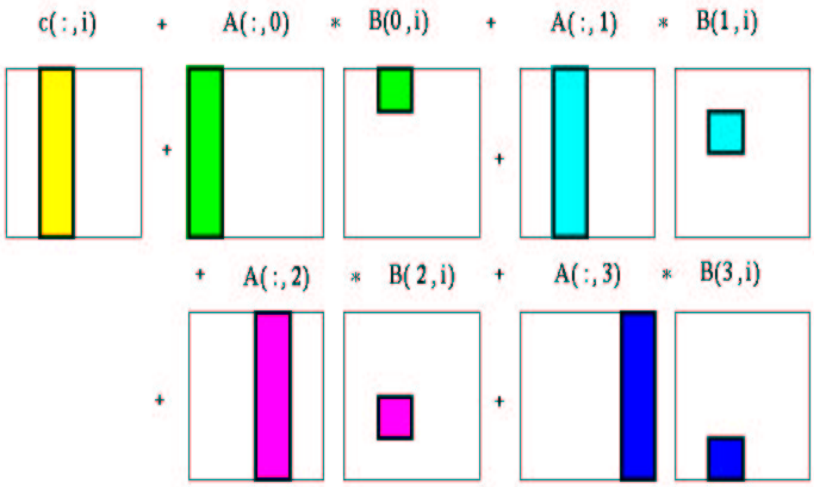


Figura 3.17: *Il blocco $i-esimo$ della matrice C è dato dalla somma dei prodotti tra i blocchi della matrice A e il blocco $i-esimo$ della matrice B, partizionato in p sottoblocchi B_{ji}.*

Per determinare C_i, ogni processore deve partizionare il proprio blocco B_i in p blocchi riga ottenendo:

$$B_{ji} \in \Re^{\frac{h}{p} \times \frac{n}{p}} \quad con \quad j = 0, \ldots, p-1$$

Con tale suddivisione si ha:

$$C_i = \sum_{k=0}^{p-1} A_k \cdot B_{ki} \quad con \quad i = 0, \ldots, p-1$$

In Fig. 3.17 è schematizzato l'algoritmo per il blocco $i = 1$.

Nel caso in cui $p = 2$, suddividiamo A e B in due blocchi di colonne nel seguente modo:

$$[A_0, A_1] \cdot [B_0, B_1] = [C_0, C_1]$$

con A_0 e $A_1 \in \Re^{m \times \frac{h}{2}}$, B_0 e $B_1 \in \Re^{h \times \frac{n}{2}}$, e le matrici C_0 e $C_1 \in \Re^{m \times \frac{n}{2}}$. Ad ogni processore viene assegnato un blocco di A e un blocco di

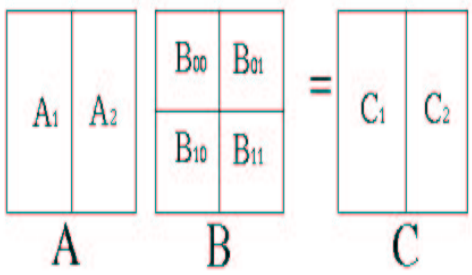


Figura 3.18: *III strategia: la matrice A viene suddivisa in due blocchi di colonne e la matrice B viene suddivisa in due blocchi di colonne. Ogni processore P_i, con $i = 0, p-1$, partiziona il proprio blocco B_i in sottoblocchi B_{ji}, con $j = 0, p-1$. Con questo tipo di suddivisione dei dati ciascun processore potrà calcolare un blocco di colonne della matrice C.*

B. In particolare:

processore P_0	**processore** P_1
A_0, B_0	A_1, B_1

Più precisamente, il processore P_0 possiede gli elementi di A_0:

$$A_0 \equiv (a_{i,j}) \quad i = 0, \ldots, 5;\ j = 0, \ldots, 2$$

e quelli di B_0:

$$B_0 \equiv (b_{i,j}) \quad i = 0, \ldots, 5;\ j = 0, \ldots, 2$$

Il processore P_1 possiede gli elementi di A_1:

$$A_1 \equiv (a_{i,j}) \quad i = 0, \ldots, 5;\ j = 3, \ldots, 5$$

e quelli di B_1:

$$B_1 \equiv b_{i,j} \quad i = 0, \ldots, 5;\ j = 3, \ldots, 5$$

Ogni processore P_i partiziona la matrice B_i in p blocchi riga, ottenendo:

$$B_i = \begin{bmatrix} B_{0i} \\ \vdots \\ B_{(p-1)i} \end{bmatrix}$$

Con i dati distribuiti e partizionati cosí ogni processore calcola gli elementi seguenti:

- P_0 calcola $c_{ij} = \sum_{k=0,1,2} a_{ik} b_{kj}$ per $i = 0, \ldots, 5$ e $j = 0, 1, 2$ (P_0 avrà una somma parziale di tutti gli elementi delle prime tre colonne della matrice C ovvero calcola: $C_0 = A_0 \cdot B_{00}$);
- P_1 calcola $c_{ij} = \sum_{k=3,4,5} a_{ik} b_{kj}$ per $i = 0, \ldots, 5$ e $j = 3, 4, 5$ (P_1 avrà una somma parziale di tutti gli elementi delle seconde tre colonne della matrice C ovvero calcola: $C_1 = A_1 \cdot B_{11}$).

Algoritmo processore P_0

```
⋮
for i = 0 to 5 do
    for j=0 to 2 do
        for k=0 to 2 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i = 0 to 5 do
    for j=3 to 5 do
        for k=3 to 5 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Procedura 3.25 - Algoritmi per il calcolo delle componenti parziali della matrice C.

Per poter calcolare il resto delle somme parziali i processori devono scambiare tra loro i blocchi di A:

processore P_0	**processore P_1**
A_1, B_0	A_0, B_1

Il processore P_0 possiede gli elementi di A_1:

$$A_1 \equiv (a_{i,j}) \quad i = 0, \ldots, 5;\ j = 3, \ldots, 5$$

Il processore P_1 possiede gli elementi di A_0:

$$A_0 \equiv (a_{i,j}) \quad i = 0, \ldots, 5;\ j = 0, \ldots, 2$$

Al secondo passo i due processori calcolano i seguenti elementi:

- P_0 calcola $c_{ij} = \sum_{k=3,4,5} a_{ik}b_{kj}$ per $i = 0, \ldots, 5$ e $j = 0, 1, 2$ (P_0 avrà le prime tre colonne della matrice C avendo calcolato: $C_0 = C_0 + A_1 \cdot B_{10} = A_0 \cdot B_{00} + A_1 \cdot B_{10}$);
- P_1 calcola $c_{ij} = \sum_{k=0,1,2} a_{ik}b_{kj}$ per $i = 0, \ldots, 5$ e $j = 3, 4, 5$ (P_1 avrà le seconde tre colonne della matrice C avendo calcolato: $C_1 = C_1 + A_0 \cdot B_{01} = A_1 \cdot B_{11} + A_0 \cdot B_{01}$).

Algoritmo processore P_0

```
⋮
for i = 0 to 5 do
    for j=0 to 2 do
        for k=3 to 5 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Algoritmo processore P_1

```
⋮
for i = 0 to 5 do
    for j=3 to 5 do
        for k=0 to 2 do
            c_ij := c_ij + a_ik · b_kj
        endfor
    endfor
endfor
⋮
```

Procedura 3.26 - Algoritmi per il calcolo delle componenti risultanti della matrice C.

Dalla Fig. 3.17 risulta chiaro che per ottenere il risultato finale è necessario ad ogni passo scambiare i blocchi della matrice A. L'idea alla base di questa strategia è far calcolare ad ogni processore tutte le componenti della matrice C appartenenti alla colonna di indice

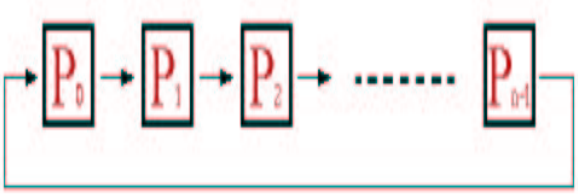


Figura 3.19: *I processori sono collegati in modo tale che ognuno di essi può comunicare con il processore che si trova alla sua destra. L'ultimo processore comunica con il primo. Tale tipo di topologia è chiamata ad anello ed è stata illustrata nel terzo capitolo.*

id con:

$$C(:, id) = C(:, id) + A(:, id) * B(id, id)$$

avendo indicato con id l'identificativo del processore, quindi ogni processore invia il proprio blocco di colonne di A al processore che si trova alla sua destra.

Tale strategia prende il nome di *1D-Systolic*[13]. Una strategia di comunicazione efficiente dei dati tra i processori è quella realizzata immaginando i processori collegati secondo una topologia ad anello (Fig. 3.19) in quanto ogni processore deve inviare al processore alla sua destra un blocco della matrice A. L'ultimo processore comunica con il primo. In questo caso si dice che l'anello è *"periodico"*.
L'algoritmo è il seguente:

[13]Tali algoritmi sono detti *sistolici* perché il flusso di dati, essendo sempre nella stessa direzione durante le fasi di comunicazione, ricorda il flusso di sangue dopo un movimento di contrazione del cuore, movimento che viene appunto detto "sistole".

Algoritmo MATMAT 1D-Systolic (III strategia m=h=n)

```
begin
  % calcolo del numero di processori p e del proprio identificativo myid
  % k=numero di colonne distribuito a ciascun processore
  k := n/p
  % k1=indice della prima colonna distribuita ad ogni processore
  k1 := k · myid
  % k2=indice dell'ultima colonna distribuita ad ogni processore
  k2 := k1 + k − 1
  % distribuzione dei dati
  A_loc := A(:, k1 : k2)
  B_loc := B(:, k1 : k2)
  C_loc := C(:, k1 : k2)
  for i=0 to k-1 do
      for j=0 to k-1 do
          C_loc(i, j) := 0
      endfor
  endfor
  % fase di calcolo
  l := k · myid
  C_loc := C_loc + A_loc B_loc(l : l + k, :)
  for i=0 to p-2 do
  % idsend: identificativo del processore a cui inviare
      idsend := myid + 1
      if (idsend > (p − 1)) then
          idsend := 0
      endif
  % idrecv: identificativo del processore da cui ricevere
      idrecv = myid − 1
      if (idrecv < 0)) then
          idrecv = p − 1
      endif
  % fase di comunicazione
      send(A_loc, idsend)
      recv(A_loc, idrecv)
  % fase di calcolo
      l := mod(l − k, n)
      C_loc := C_loc + A_loc B_loc(l : l + k, :)
  endfor
end
```

Procedura 3.27 - Algoritmo 1D-Systolic per il calcolo del prodotto matrice per matrice

L'algoritmo 1D-Systolic è caratterizzato da una distribuzione dei dati e da uno schema di comunicazione monodimensionale.

IV Strategia

Supponiamo di avere $p = 9$ processori. Immaginiamo inoltre che i processori siano disposti secondo una topologia a griglia cartesiana (vedi paragrafo 2.2) e distribuiamo i blocchi nel modo seguente:

P_{00} $A_{00}B_{00}$	P_{01} $A_{01}B_{11}$	P_{02} $A_{02}B_{22}$
P_{10} $A_{11}B_{10}$	P_{11} $A_{12}B_{21}$	P_{12} $A_{10}B_{02}$
P_{20} $A_{22}B_{20}$	P_{21} $A_{20}B_{01}$	P_{22} $A_{21}B_{12}$

avendo indicato con $P_{i,j}$ il processore di riga i_{esima} e colonna j_{esima}. In questo caso si è suddiviso A e B in p blocchi di righe e in p blocchi di colonne, ottenendo cosí p^2 blocchi di A e di B:

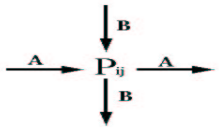
$$\begin{bmatrix} A_{00} & \dots & A_{0,p-1} \\ \vdots & & \vdots \\ A_{p-1,0} & \dots & A_{p-1,p-1} \end{bmatrix} \cdot \begin{bmatrix} B_{00} & \dots & B_{0,p-1} \\ \vdots & & \vdots \\ B_{p-1,0} & \dots & B_{p-1,p-1} \end{bmatrix} = \begin{bmatrix} C_{00} & \dots & C_{0,p-1} \\ \vdots & & \vdots \\ C_{p-1,0} & \dots & C_{p-1,p-1} \end{bmatrix}$$

e il prodotto $A \cdot B = C$ viene decomposto in prodotti $A_{ik}B_{kj}$. Al processore P_{ij} vengono assegnati i blocchi A_{ik} e B_{kj} con:

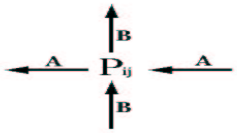
$$k =< i + j >_p = resto(i + j, p)$$

Con questa distribuzione delle matrici, ogni processore può effettuare solo il calcolo di $C_{ij} = C_{ij} + A_{ik}B_{kj}$, cioè ha somme parziali del blocco C_{ij} di C. Per completare il calcolo ogni processore, quindi,

ha necessità di acquisire i blocchi di A e di B. Si può pensare di far comunicare ogni processore con i quattro processori ad esso vicini utilizzando una topologia di tipo toroidale (vedi paragrafo 2.3), secondo lo schema di comunicazione seguente:



O, equivalentemente, è possibile utilizzare sempre in una topologia di tipo toroidale quest'altro schema di comunicazione:



Scegliamo ad esempio quest'ultimo schema di comunicazione per illustrare la IV strategia. Ad ogni passo ogni processore su di una stessa riga invia al processore che si trova nella colonna alla sua sinistra nella griglia un blocco della matrice A e riceve dal processore che si trova nella colonna alla sua destra nella griglia un blocco della matrice A. Inoltre ogni processore su di una stessa colonna invia al processore che si trova sulla riga superiore della griglia un blocco della matrice B e riceve dal processore che si trova sulla riga inferiore della griglia un blocco della matrice B.

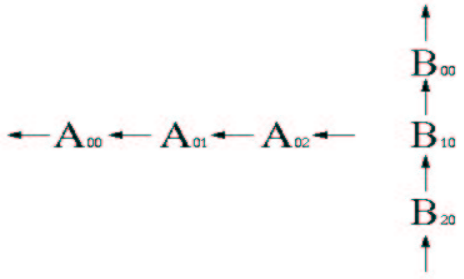
- **I PASSO**

  Sulla riga *i-esima* della griglia di processori viene assegnato a P_{ij}, con $j = 0, 1, 2$, il blocco $A_{i,mod(i+j,3)}$ della matrice A e il blocco $B_{mod(i+j,3),j}$ della matrice B:

P_{00} $C_{00} = C_{00} + A_{00}B_{00}$	P_{01} $C_{01} = C_{01} + A_{01}B_{11}$	P_{02} $C_{02} = C_{02} + A_{02}B_{22}$
P_{10} $C_{10} = C_{10} + A_{11}B_{10}$	P_{11} $C_{11} = C_{11} + A_{12}B_{21}$	P_{12} $C_{12} = C_{12} + A_{10}B_{02}$
P_{20} $C_{20} = C_{20} + A_{22}B_{20}$	P_{21} $C_{21} = C_{21} + A_{20}B_{01}$	P_{22} $C_{22} = C_{22} + A_{21}B_{12}$

- **II PASSO**

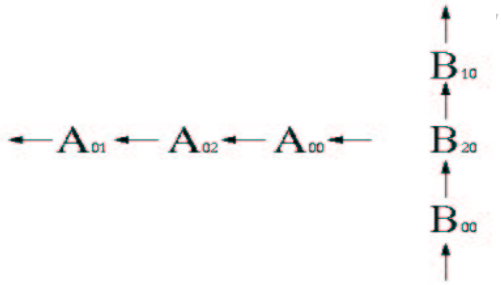
  Sulla riga *i-esima* della griglia viene inviato ad ogni processore P_{ij} il blocco $A_{i,mod(i+j+passo-1,3)}$ della matrice A dal procesore $P_{i,mod(j+1,p)}$ posto nella colonna successiva della griglia cartesiana. Su ogni colonna di processori P_{ij} invia il suo blocco di B al processore $P_{mod(i-1,p),j}$, appartenente alla riga precedente della griglia. Ad esempio, se consideriamo la prima riga e la prima colonna, le comunicazioni possono essere schematizzate nel modo seguente:



P_{00} $C_{00} = C_{00} + A_{01}B_{10}$	P_{01} $C_{01} = C_{01} + A_{02}B_{21}$	P_{02} $C_{02} = C_{02} + A_{00}B_{02}$
P_{10} $C_{10} = C_{10} + A_{12}B_{20}$	P_{11} $C_{11} = C_{11} + A_{10}B_{01}$	P_{12} $C_{12} = C_{12} + A_{11}B_{12}$
P_{20} $C_{20} = C_{20} + A_{20}B_{00}$	P_{21} $C_{21} = C_{21} + A_{21}B_{11}$	P_{22} $C_{22} = C_{22} + A_{22}B_{22}$

- **III PASSO**

  Sulla riga *i-esima* di processori viene inviato ad ogni processore P_{ij} il blocco $A_{i,mod(i+j+passo-1,3)}$ della matrice A dal processore $P_{i,mod(j+1,p)}$ posto nella colonna successiva nella griglia cartesiana. Su ogni colonna di processori P_{ij} invia il suo blocco di B al processore $P_{mod(i-1,p),j}$ appartenente alla riga precedente della griglia. Ad esempio, se consideriamo la prima riga e la prima colonna, le comunicazioni sono schematizzate nel modo seguente:



P_{00} $C_{00} = C_{00} + A_{02}B_{20}$	P_{01} $C_{01} = C_{01} + A_{00}B_{01}$	P_{02} $C_{02} = C_{02} + A_{01}B_{12}$
P_{10} $C_{10} = C_{10} + A_{10}B_{00}$	P_{11} $C_{11} = C_{11} + A_{11}B_{11}$	P_{12} $C_{12} = C_{12} + A_{12}B_{22}$
P_{20} $C_{20} = C_{20} + A_{21}B_{10}$	P_{21} $C_{21} = C_{21} + A_{22}B_{21}$	P_{22} $C_{22} = C_{22} + A_{20}B_{02}$

Tale strategia, essendo il flusso di dati lungo due direzioni ortogonali, e avendo lungo ciascuna direzione le stesse caratteristiche di quella *1D-Systolic*, prende il nome di *2D-Systolic*. Di seguito è riportato l'algoritmo. Sono stati denotati con north, south, west e east rispettivamente l'identificativo del processore a cui inviare i blocchi della matrice B, del processore da cui ricevere i blocchi della matrice B, del processore a cui inviare i blocchi della matrice A, e del processore da cui ricevere i blocchi della matrice A.

Algoritmo MATMAT 2D-Systolic (IV strategia m=h=n)

```
begin
  % calcolo del numero di processori p^2 e del proprio identificativo
  % k=numero di righe e di colonne distribuite a ciascun processore
   k := n/p
  % k1x=prima riga di A distribuita a (idx, idy)
   k1x := idx · k
  % k1y=prima colonna di A e prima riga di B distribuite a (idx, idy)
   k1y := mod(idx + idy, p) · k
  % k2x=ultima riga di A distribuita a (idx, idy)
   k2x := k1x + k
  % k2y=ultima colonna di A ed ultima riga di B distribuite a (idx, idy)
   k2y := k1y + k
  % k1z=prima colonna di B distribuita a (idx, idy)
   k1z := idy · k
  % k2z=ultima colonna di B distribuita a (idx, idy)
   k2z := k1z + k
  % distribuzione dei dati
   A_loc := A(k1x : k2x, k1y : k2y)
   B_loc := B(k1y : k2y, k1z : k2z)
   C_loc := C(k1x : k2x, k1z : k2z)
  % calcolo dell'identificativo dei processi con cui c'è comunicazione
   ovest=(idx,mod(idy − 1, p))
   east=(idx,mod(idy + 1, p))
   south=(mod(idx + 1, p), idy)
   north=(mod(idx − 1, p), idy)
  % fase di calcolo
       C_loc = C_loc + A_loc B_loc
   for i=0 to p-2 do
  % fase di comunicazione
       send(A_loc, west) ; recv(A_loc, east)
       send(B_loc, north) ; recv(B_loc, south)
  % fase di calcolo
       C_loc = C_loc + A_loc B_loc
   endfor
end
```

Procedura 3.28 - Algoritmo 2D-Systolic per il calcolo del prodotto matrice per matrice.

V Strategia

Come nella strategia precedente, si possono suddividere le matrici A e B in p^2 blocchi e assegnare al processore P_{ij} le sottomatrici A_{ij} e B_{ij}. Supponiamo di avere un numero di processori $p = 9$. Immaginiamo, inoltre, che i processori siano disposti secondo una griglia cartesiana e distribuiamo i blocchi nel modo seguente:

P_{00} $A_{00}B_{00}$	P_{01} $A_{01}B_{01}$	P_{02} $A_{02}B_{02}$
P_{10} $A_{10}B_{10}$	P_{11} $A_{11}B_{11}$	P_{12} $A_{12}B_{12}$
P_{20} $A_{20}B_{20}$	P_{21} $A_{21}B_{21}$	P_{22} $A_{22}B_{22}$

In questo caso si è suddiviso A e B in p blocchi di righe e in p blocchi di colonne, ottenendo cosí p^2 blocchi:

$$\begin{bmatrix} A_{00} & \dots & A_{0,p-1} \\ \vdots & & \vdots \\ A_{p-1,0} & \dots & A_{p-1,p-1} \end{bmatrix} \cdot \begin{bmatrix} B_{00} & \dots & B_{0,p-1} \\ \vdots & & \vdots \\ B_{p-1,0} & \dots & B_{p-1,p-1} \end{bmatrix} = \begin{bmatrix} C_{00} & \dots & C_{0,p-1} \\ \vdots & & \vdots \\ C_{p-1,0} & \dots & C_{p-1,p-1} \end{bmatrix}$$

e il prodotto $A \cdot B = C$ viene decomposto in prodotti $A_{ik}B_{kj}$. Al processore P_{ij} vengono assegnati i blocchi A_{ij} e B_{ij}. Ciascun processore calcola un blocco della matrice C nel modo seguente:

$$\begin{aligned}
P_{00} \rightarrow C_{00} &= A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20} \\
P_{01} \rightarrow C_{01} &= A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21} \\
P_{02} \rightarrow C_{02} &= A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22} \\
P_{10} \rightarrow C_{10} &= A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20} \\
P_{11} \rightarrow C_{11} &= A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21} \\
P_{12} \rightarrow C_{12} &= A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22} \\
P_{20} \rightarrow C_{20} &= A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20}
\end{aligned}$$

$$P_{21} \rightarrow C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21}$$
$$P_{22} \rightarrow C_{22} = A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22}$$

Per poter completare il calcolo dei blocchi C_{ij}, ad ogni passo a partire dai processori diagonali, ogni processore invia il proprio blocco di A a tutti i processori della propria riga, ed un blocco di B al processore della riga precedente e della medesima colonna.

- **I PASSO**

  Sulla riga $i-esima$ della griglia di processori viene inviato a tutti i processori il blocco $A_{i,i}$ della matrice A. Ad esempio, sulla riga 0 viene inviato a tutti i processori il blocco A_{00} della matrice A, sulla riga 1 viene inviato a tutti i processori il blocco A_{11} della matrice A e sulla riga 2 viene inviato a tutti i processori il blocco A_{22} della matrice A. Tutti i processori utilizzano il blocco B_{ij} della matrice B a loro assegnato inizialmente. Nella figura sono evidenziati in rosso i prodotti che ciascun processore puó calcolare.

P_{00} $C_{00} = A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}$	P_{01} $C_{01} = A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$	P_{02} $C_{02} = A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22}$
P_{10} $C_{10} = A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20}$	P_{11} $C_{11} = A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}$	P_{12} $C_{12} = A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22}$
P_{20} $C_{20} = A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20}$	P_{21} $C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21}$	P_{22} $C_{12} = A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22}$

- **II PASSO**

  Sulla riga i-$esima$ di processori viene inviato a tutti i processori il blocco $A_{i,mod(i+passo-1,3)}$ della matrice A. Ad esempio, sulla riga 0 viene inviato a tutti i processori il blocco A_{01} della matrice A. Su ogni colonna di processori, ogni processore invia al processore della riga precedente il suo blocco di B e riceve

dal processore della riga successiva il blocco di B. Nella figura sono evidenziati in rosso i prodotti che ciascun processore può calcolare.

P_{00} $C_{00} = A_{00}B_{00}+A_{01}B_{10}+A_{02}B_{20}$	P_{01} $C_{01} = A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$	P_{02} $C_{02} = A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22}$
P_{10} $C_{10} = A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20}$	P_{11} $C_{11} = A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}$	P_{12} $C_{12} = A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22}$
P_{20} $C_{20} = A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20}$	P_{21} $C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21}$	P_{22} $C_{12} = A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22}$

- **III PASSO**

  Sulla riga i-esima di processori viene inviato a tutti i processori il blocco $A_{i,mod(i+passo-1,3)}$ della matrice A. Ad esempio sulla riga 0 viene inviato a tutti i processori il blocco A_{02} della matrice A. Su ogni colonna di processori, ogni processore invia al processore della riga precedente il suo blocco di B e riceve dal processore della riga successiva il blocco di B. Nella figura sono evidenziati in rosso i prodotti che ciascun processore può calcolare.

P_{00} $C_{00} = A_{00}B_{00} + A_{01}B_{10}+A_{02}B_{20}$	P_{01} $C_{01} = A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$	P_{02} $C_{02} = A_{00}B_{02} + A_{01}B_{12} + A_{02}B_{22}$
P_{10} $C_{10} = A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20}$	P_{11} $C_{11} = A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}$	P_{12} $C_{12} = A_{10}B_{02} + A_{11}B_{12} + A_{12}B_{22}$
P_{20} $C_{20} = A_{20}B_{00} + A_{21}B_{10} + A_{22}B_{20}$	P_{21} $C_{21} = A_{20}B_{01} + A_{21}B_{11} + A_{22}B_{21}$	P_{22} $C_{12} = A_{20}B_{02} + A_{21}B_{12} + A_{22}B_{22}$

Tale strategia prende il nome di *BMR* (Broadcast Multiply Rolling)[14] e l'algoritmo è il seguente:

[14] Una strategia simile viene utilizzata anche in PBLAS, la libreria parallela di algebra lineare di cui si parlerà nel capitolo 6. La differenza risiede nel fatto che i blocchi della matrice B vengono spediti al processore della riga successiva della colonna medesima.

Algoritmo MATMAT BMR (V strategia m=h=n)

```
begin
  % calcolo del numero di processori p^2 e del proprio identificativo (idx, idy)
  % del proprio identificativo (idx, idy)
  % k=numero di righe e di colonne distribuite
  % a ciascun processore
  k := n/p
  % k1x=prima riga di A e B distribuite a (idx, idy)
  k1x := idx · k
  % k2x=ultima riga di A e B distribuite a (idx, idy)
  k2x := k1x + k
  % k1y=prima colonna di A e B distribuite a (idx, idy)
  k1y := idy · k
  % k2y=ultima colonna di A e B distribuite a (idx, idy)
  k2zy = k1y + k
  % distribuzione dei dati
  % inizializzazione delle varibili locali
  A_loc := A(k1x : k2x, k1y : k2y)
  B_loc := B(k1x : k2x, k1y : k2y)
  C_loc := C(k1x : k2x, k1y : k2y)
  % calcolo dei processori con cui c'è comunicazione
  north:=(idx,mod(idy − 1, p))
  south:=(idx,mod(idy + 1, p))
  for i=0 to p-1 do
  % prima fase di comunicazione: broadcast sulle righe;
  % il processore, dei sottocontesti riga della griglia,
  % di identificativo proc comunica con tutti i processori
  % che si trovano sulla sua stessa riga
      proc := mod(idx + i, p)
      sendbroadcast(A_loc, row, proc)
      recvbroadcast(A_temp, row, proc)
  % fase di calcolo
      C_loc := C_loc + A_temp B_loc
  % seconda fase di comunicazione
      send(B_loc, north)
      recv(B_loc, south)
  endfor
end
```

Procedura 3.29 - Algoritmo per eseguire il prodotto matrice per matrice secondo lo schema BMR.

Stima teorica delle prestazioni

Valutiamo le prestazione degli algoritmi 1D-Systolic, 2D-Systolic e BMR su un calcolatore MIMD a memoria distribuita costituito da $nproc = p \times p$ processori.

Siano

- $A, B, C \in \Re^{n \times n}$;
- t_{com} il tempo per la comunicazione di un dato floating point tra due processori;
- t_{broad} il tempo per la comunicazione di un dato floating point in un broadcast orizzontale;
- t_{calc} il tempo per l'esecuzione di una operazione floating point;
- T_{calc} il tempo totale di esecuzione;
- T_{com} il tempo totale di comunicazione.

Nell'algoritmo 1D-Systolic, ogni processore P_i, $0 \leq i \leq p-1$, effettua il prodotto:

$$A_i \cdot B_i \tag{3.1}$$

con $A_i \in \Re^{N \times \frac{N}{p}}$, $B_i \in \Re^{\frac{N}{p} \times \frac{N}{p}}$.
L'operazione (3.1) richiede:

$$T_{calc} = 2 \cdot \frac{N^3}{p^2}\, t_{calc}$$

e viene eseguita p volte. In definitiva:

$$T_{calc} = p \cdot \frac{2N^3}{p^2}\, t_{calc} = \frac{2N^3}{p}\, t_{calc}$$

Inoltre, al passo j, con $j = 0, \ldots, p-2$, ogni processore P_i deve inviare il proprio blocco di matrice A:

$$A_i \in \Re^{N \times \frac{N}{p}}$$

al processore $P_{mod(i+1,p)}$. Il tempo di comunicazione è quindi:

$$T_{com} = N \cdot \frac{N}{p}\, t_{com} = \frac{N^2}{p}\, t_{com}$$

Tale spedizione viene eseguita $p-1$ volte, e quindi:

$$T_{com} = \frac{N^2}{p} \cdot (p-1)\, t_{com} = \frac{N^2(p-1)}{p}\, t_{com}$$

Nell'algoritmo 2D-Systolic, ogni processore $P_{i,j}$, $0 \leq i \leq p-1$ e $0 \leq j \leq p-1$, effettua il prodotto:

$$A_{ik} \cdot B_{kj} \tag{3.2}$$

con A_{ik} e $B_{kj} \in \Re^{\frac{N}{p} \times \frac{N}{p}}$.
L'operazione (3.2) richiede:

$$T_{calc} = 2 \cdot \frac{N^3}{p^3}\, t_{calc}$$

e viene eseguita per p volte. In definitiva:

$$T_{calc} = p \cdot \frac{2N^3}{p^3}\, t_{calc} = \frac{2N^3}{p^2}\, t_{calc}$$

Inoltre, ad ogni passo h, con $h = 0, \ldots, p-2$, ogni processore P_{ij} deve inviare il proprio blocco di matrice A:

$$A_{ik} \in \Re^{\frac{N}{p} \times \frac{N}{p}}$$

al processore $P_{i,mod(j-1,p)}$ e il proprio blocco di matrice B:

$$B_{kj} \in \Re^{\frac{N}{p} \times \frac{N}{p}}$$

al processore $P_{mod(i-1,p),j}$. Il tempo di comunicazione è quindi:

$$T_{com} = 2 \cdot \frac{N}{p} \cdot \frac{N}{p} \, t_{com} = \frac{2N^2}{p^2} \, t_{com}$$

Tale spedizione viene eseguita $p-1$ volte. In definitiva:

$$T_{com} = \frac{2N^2}{p^2} \cdot (p-1) \, t_{com} = \frac{2N^2(p-1)}{p^2} \, t_{com}$$

Infine, nell'algoritmo BMR si ha una suddivisione dei blocchi di matrice simile all'algoritmo 2D-Systolic, cosí da avere lo stesso numero di operazioni fl. point:

$$T_{calc} = p \cdot \frac{2N^3}{p^3} \, t_{calc} = \frac{2N^3}{p^2} \, t_{calc}$$

Inoltre, ad ogni passo h, con $h = 0, \ldots, p-1$ il processore $P_{mod(h+i,p)}$ dei sottocontesti riga della griglia[15] deve inviare il proprio blocco di matrice A:

$$A_{ik} \in \Re^{\frac{N}{p} \times \frac{N}{p}}$$

a tutti i processi appartenenti al proprio sottocontesto mediante un broadcast. Il tempo di comunicazione è quindi:

$$T_{broad} = \frac{N}{p} \cdot \frac{N}{p} \, t_{broad} = \frac{N^2}{p^2} \, t_{broad}$$

Tale spedizione viene eseguita p volte.

[15] Costruita una griglia cartesiana bidimensionale, di dimensione $p \times p$, si definisce sottocontesto riga l'insieme dei processori disposti lungo una delle p righe della griglia. In modo del tutto analogo si definiscono i sottocontesti colonna della griglia.

In definitiva:

$$T_{broad} = \frac{N^2}{p^2} \cdot p\, t_{broad} = \frac{N^2}{p}\, t_{broad}$$

Inoltre ogni processore P_{ij} deve inviare il proprio blocco di matrice B:

$$B_{kj} \in \Re^{\frac{N}{p} \times \frac{N}{p}}$$

al processore $P_{mod(i-1,p),j}$. Il tempo di comunicazione è :

$$T_{com} = \frac{N}{p} \cdot \frac{N}{p}\, t_{com} = \frac{N^2}{p^2}\, t_{com}$$

Tale spedizione viene eseguita $p-1$ volte. In definitiva:

$$T_{com} = \frac{N^2}{p^2} \cdot (p-1)\, t_{com} = \frac{N^2(p-1)}{p^2}\, t_{com}$$

In conclusione, risulta:

Per la strategia 1D-Systolic:

$$T_{1D} \simeq \frac{2N^3}{p} T_{calc} + \frac{N^2(p-1)}{p} T_{com}$$

Per la strategia 2D-Systolic:

$$T_{2D} \simeq \frac{2N^3}{p^2} T_{calc} + \frac{2N^2(p-1)}{p^2} T_{com}$$

Per la strategia BMR:

$$T_{BMR} \simeq \frac{2N^3}{p^2} T_{calc} + \frac{N^2(p-1)}{p^2} T_{com} + \frac{N^2}{p} T_{broad}$$

Da cui segue

$$T_{1D} \geq T_{2D} \ \ per \ \ p > 2$$

e

$$T_{BMR} \leq T_{2D} \leq T_{1D} \ \ se \ \ t_{broad} \leq t_{com}$$

Dunque il metodo BMR risulta piú efficiente dei metodi sistolici se il tempo di broadcast è inferiore al tempo di comunicazione tra i processori.

Da osservare inoltre che per tutte e tre le strategie la complessità computazionale è dell'ordine di $O(N^3)$ mentre il costo delle comunicazione è dell'ordine di $O(N^2)$. Il costo delle comunicazioni cresce quindi piú lentamente del costo computazionale (effetto superficie-volume):

$$\frac{T_{com}}{T_{calc}} = O\left(\frac{1}{N}\right)$$