

1. Si consideri il prodotto matrice per matrice  $A \cdot B = C$ , con  $A \in \mathbb{R}^{128 \times 96}$ ,  $B \in \mathbb{R}^{96 \times 64}$ , su un'architettura MIMD-DM, costituita da 16 processori. La matrice A è distribuita per righe, mentre la matrice B è distribuita per colonne.

a. Con i dati così distribuiti cosa possono calcolare i singoli processori?

### Strategy 1 → solo processo deve avere il risultato

La strategy è di distribuire la matrice A in blocchi di righe e la matrice B in blocchi di colonne. Ogni processo riceve  $A_{loc} \in \mathbb{R}^{8 \times 96}$  e  $B_{loc} \in \mathbb{R}^{36 \times 3}$ . Ogni processo può calcolare quindi  $C_{loc}$ , ovvero il prodotto  $T_{loc}$   $A_{loc} \cdot B_{loc} = C_{loc} \in \mathbb{R}^{8 \times 3}$ . Una volta calcolato il prodotto ogni processo invia a p-1 processi il proprio blocco B, quindi il proprio  $B_{loc}$ , ed infine ogni processo calcola il prodotto con tutti i  $B_{loc}$  degli altri processori ed il suo  $A_{loc}$ . Ogni processo esegue quindi  $16 \times (8 \times 3 \times (36 \text{ moltiplicazioni} + 35 \text{ somme}))$

### Definizione classica

$$T(1) = 128 \times 96 \times (36 \text{ moltiplicazioni} + 35 \text{ somme}) = 1143504 \text{ TCACC}$$

$$T(16) = T_{ACC} + T_{COM}. \quad T_{ACC} = 16 \times (8 \times 3 \times (36 + 35)) = 73344 \text{ TCACC}$$

$T_{COM}$  non, dobbiamo considerare i p-1 m-1 processi del proprio blocco  $B_{loc}$ , e poi ogni processo invia a p-1 la matrice ottenuta, quindi:

$$T_{COM} = \underbrace{15 \times (36 \times 3)}_{\text{matrice } B_{loc}} + \underbrace{35 \times (16 \times (p \times 3))}_{\text{matrice } P, \text{ moltiplicazione}} = 6320 + 5760 = 60800 \text{ TCACC} \quad T_{COM} = 2TCACC$$

$$T(16) = 73344 + 60800 = 93504 \text{ TCACC}$$

$$\Theta_{HT} = (P \times T(P)) - T(1) = 16 \times 93504 - 1143504 = 32280 \quad E(16) = \frac{T(16)}{16} = \frac{1228}{16} = 0,78$$

$$\Theta_{HC} = \frac{T_{COM}}{T_{ACC}} = \frac{100800}{73344 \text{ TCACC}} = \frac{100800}{73344} = \frac{105}{382} = 0,28$$

Work generalizate e lavoro parallelo. non posso essere calcolato poiché tutto è parallellizzato e quindi  $\alpha = 0 \quad 1-\alpha = \frac{1}{16} \quad S(16) = 16 \quad E(16) = 1$

### Strategy 2 → si richiede che 1 solo processo riceva il risultato

La matrice A viene distribuita per blocchi di colonne mentre la matrice B non è distribuita per blocchi di righe.  $A_{loc} \in \mathbb{R}^{128 \times 6}$  e  $B_{loc} \in \mathbb{R}^{6 \times 64}$ . Ogni processo calcola quindi  $C_{loc} \in \mathbb{R}^{128 \times 6} = A_{loc} \times B_{loc}$ . Una volta terminato, per ottenere il risultato finale bisogna sommare tutti i  $C_{loc}$  del tutto, processi, affinché si ottenga la matrice finale  $C \in \mathbb{R}^{128 \times 64}$ .

### Strategy 1 - Somme Definizione classica

$$T(1) = 1143504 \text{ TCACC} \quad T(16) = T_{ACC} + T_{COM}$$

$$T_{ACC} = \underbrace{128 \times 96 \times (6 \text{ moltiplicazioni} + 5 \text{ somme})}_{\text{Prodotto scalare } A_{loc} \times B_{loc}} +$$

Vengono eseguiti p-1 passi di comunicazione.

$$15 \times (128 \times 6) = 92360 + 67584 = 159944 \quad T_{COM} = 2TCACC$$

$$T_{COM} = 15 \times (128 \times 6) = 92360 \text{ TCACC} \quad T(16) = 159944 + 184320 = 344264 \text{ TCACC}$$

$$S(16) = \frac{T(1)}{T(16)} = \frac{1143504}{159944} = \frac{191}{16} = 11,43 \quad E(16) = \frac{S(16)}{16} = 0,71 \quad \Theta_{HC} = \frac{184320}{159944} = \frac{15}{13} = 1,15$$

$$Oht = 16 \times 34064 - 1143504 = 6335520$$

Meno generalizzata

Worst = meno generalizzata, poiché moltiplicazioni parallele di matrice.

$$\alpha_3 = \frac{15 \times 128 \times 6}{1143504 \cdot 1143504} \alpha_{36} = \frac{16 \times 128 \times 6 \times (6+5)}{1143504} = \frac{1083364}{1143504}$$

$$S(36) < \frac{1}{\frac{92960}{1143504} + \frac{1083364}{1143504}} = \frac{1}{\frac{15}{193} + \frac{11}{193}} = \frac{1}{\frac{26}{193}} = 7,34 \quad C(36) = \frac{7,34}{16} < 0,45$$

### Strategy 2 Difinizione Chiarice

$$T(S) = 1143504 T_{ACC} \quad T(36) = T_{ACC} + T_{COM}$$

$$T_{ACC} = \underbrace{128 \times 6 \times (6 \text{ moltiplicazione} + 5 \text{ somme})}_{\text{Problema scalare } A_{loc} \times B_{loc}} + \underbrace{4 \times (128 \times 6)}_{\text{Somme tra matr.}} = 24576 + 67584 = 92360 \quad T_{COM} = 2T_{ACC}$$

$$T_{COM} = 6 \times (128 \times 6) = 24576 \quad T(36) = 92360 + 49152 = 141512 \quad T_{CALC}$$

$$S(16) = \frac{T(1)}{T(36)} = \frac{1143504}{141512} = \frac{191}{23} = 8,3 \quad C(36) = \frac{S(36)}{36} = 0,52 \quad Ohc = \frac{49152}{92360} = \frac{8}{15} = 0,53$$

$$Oht = 16 \times 161392 - 1143504 = 5087488$$

Meno generalizzata

$$\alpha_0 = \frac{6344}{1143504} \quad \alpha_1 = \frac{128 \times 6}{1143504} \quad \alpha_2 = \frac{2 \times (128 \times 6)}{1143504} \quad \alpha_n = \frac{24576}{1143504} \quad \alpha_f = \frac{67584}{1143504} \quad \alpha_{36} = \frac{1083364}{1143504}$$

$$S(36) = \frac{1}{\frac{6344}{1143504} \times 6 + \frac{1083364}{1143504}} = \frac{191}{15} = 12,03 \quad C(36) = \frac{12,03}{36} = 0,33$$

Meno

$$\alpha = \frac{6344}{1143504} \quad 1-\alpha = \frac{1083364 + 67584 + 24576 + 12288}{1143504} = \frac{1164360}{1143504} = \frac{100}{981}$$

$$S(36) = \frac{1}{\frac{1}{193} + \frac{193}{193}} = 14,83 \quad C(36) = \frac{14,83}{36} = 0,32$$

### Strategy 3 Sintesi Strategy 2 e Strategy 3 sommando altri fattori relativi ad un solo processore, prima

Le matrice A e la Matrice B vengono suddivise in Divisione Righe + Divisione Colonne, basandosi su un comunicatore di una griglia di processori  $4 \times 4 - T_{IT}$ , per cui occorrono  $A_{loc} \in \mathbb{R}^{32 \times 24}$  e  $B_{loc} \in \mathbb{R}^{24 \times 32}$ , sostanzialmente ogni processo

dovrà calcolare una porzione della matrice finale, la quale deve essere sommata a tutte le altre porzioni di tutti i processori per ottenere il risultato finale. Ogni processo dovrà inviare il proprio blocco di A al suo processo. Sulla riga della righe, di conseguenza ogni processo dovrà inviare a tutti,

0	1	2	3	
0	$P_0$	$P_3$	$P_2$	$P_1$
1	$P_4$	$P_5$	$P_6$	$P_7$
2	$P_8$	$P_9$	$P_{10}$	$P_{11}$
3	$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$


2	$P_8$	$P_9$	$P_{10}$	$P_{11}$
3	$P_{12}$	$P_{13}$	$P_{14}$	$P_{15}$


Ri-capitolando, ogni processo calcola  $\frac{P}{h} = \frac{36}{4} = 6 \times 32 \times 32$  mat. pl. ciascuno, quindi somma i risultati delle colonne appartenenti al proprio blocco di A.

**Definizione classica**  $T(S) = 117350h$   $T(36) = T_{\text{calc}} + T_{\text{comm}}$   $T_{\text{comm}} = 2T_{\text{calc}}$

$$T_{\text{calc}} = h \times 32 \times 32 \times (2h \text{ mat. pl. ciascuna} + 23 \text{ somme}) + 3 \times 32 \times 32 = 72382 + 1152 = 73344h$$

$$T_{\text{comm}} = \underbrace{3 \times 32 \times 24}_{\text{Scambio blocco}} + \underbrace{3 \times 24 \times 32}_{\text{Scambio blocco B}} + \underbrace{15 \times 32 \times 32}_{\text{Risultato finale}} = 2304 + 864 + 5460 = 8928h$$

Se, l'risultato al un solo processo

$$T(36) = 117856 + 73344h = 93200h \quad S(36) \cdot \frac{117350h}{93200} = 12,86 \quad E(36) = 0,80$$

$$\text{Ohc} = \frac{117856}{73344h} < 0,23 \quad \text{Oht} = 16 \times 93200 - 117350h = 285636$$

senza overhead

$$T(S) = 6336 + 73344h = 79680h \quad S(S) = \frac{334350h}{79680} = 14,72 \quad E(S) = 0,92$$

$$\text{Ohc} = \frac{6336}{73344h} = 0,08 \quad \text{Oht} = 16 \times 79680 - 117350h = 103376.$$

Ware generalizzata

Ora mom si può calcolare

Nonsarà il calcolo di  $S(S)$  poiché non viene eseguito nulla in sequenziale, tutto è eseguito in parallelo, non c'è la possibilità di calcolare poiché è tutto in funzione di 36 processi, quindi avremo  $S_{36} = 3 - k = 1$  quindi  $S(36) = 16$   $E(36) = 1$  pertanto mom si può calcolare.

Speed up, efficienza

Svolgere l'esercizio con **Strategy 2** le strategie, calcolare Ware, generalizzare, definizioni

**Strategia 1 - Divisione A blocco di riga** Rispetto in 3 processi classici overhead com. overhead totale e tempo di esecuzione dell'algoritmo

Effettuare ricerca un  $A_{loc}$  di dimensione  $A_{loc} \in \mathbb{R}^{3 \times 20}$  ed ognuno ha il proprio vettore b completo.

Ogni processo calcola  $3 \times (30 \text{ mat. pl. ciascuna} + 23 \text{ somme})$ , una volta calcolato ogni processo avrà un sottovettore di C denominato  $C_{loc} \in \mathbb{R}^3$ ; tutti i processi invieranno a P0 il proprio  $C_{loc}$ .

**Definizione classica**

Il tempo di esecuzione dell'algoritmo:  $T(p) = (Ts + \frac{T(p)}{p})t_{\text{calc}} + (Tc)t_{\text{comm}}$ ,

$$T(S) = 48 \times (30 \text{ mat. pl. ciascuna} + 23 \text{ somme}) = 2832T_{\text{calc}} \quad T(36) = T_{\text{calc}} + T_{\text{comm}} \quad T_{\text{comm}} = 2T_{\text{calc}}$$

$$T_{\text{calc}} = 3 \times (30 + 23) = 129T_{\text{calc}} \quad T_{\text{comm}} = 15 \times 3 = 45T_{\text{comm}} \quad T(36) = 198 + 90 = 288T_{\text{calc}}$$

$$S(36) = \frac{T(1)}{T(36)} = \frac{2832}{267} = 10,60 \quad E(36) = \frac{10,60}{36} = 0,66 \quad \text{Ohc} = \frac{T_{\text{comm}}}{T_{\text{calc}}} = \frac{90}{267} = \frac{30}{89} = 0,50$$

$$\text{Oht} = (16 \times 267) - 2832 = 1440 \quad T(p) = \left(0 + \frac{267}{36}\right)t_{\text{calc}} + 45t_{\text{comm}} = 106,68 + t_{\text{calc}}$$

**Ware e Ware generalizzata** Tutte le operazioni sono svolte in parallelo, pertanto si calcola invece una sola calcolab  $S(36) = 36$   $E(36) = 1$

**Strategia 2 - Divisione di A per blocchi di colonna**

A differenza della prima strategia, blocchi di A ovvero  $A_{loc}$  hanno dimensioni diverse per i processi, poiché  $n_{loc} \times n_{loc}^2$

### Strategy C - Divisione di $\mathbf{A}$ per blocco di colonne

A differenza della prima strategia, blocco di A ovvero blocchi hanno dimensioni diverse per i processori, poiché 30 ovvero il numero di colonne non è divisibile per 16 allora i primi 14 processori riceveranno  $A_{LOC} \in \mathbb{R}^{6 \times 2}$  e blocchi  $\in \mathbb{R}^2$  mentre gli ultimi 2 processori riceveranno  $A_{LOC} \in \mathbb{R}^{6 \times 3}$  e blocchi  $\in \mathbb{R}^1$ . Per procurare calcolatrice un'utile che deve essere sommato con tutti gli altri utili dei processori. Si eseguono le due strategie delle somme.

### Strategy D - Somma sequenziale

Wore generalizzata = Wore in questo caso

Definizione Classica

$$T_{COM} = 2T_{CALC}$$

$$T(3) = 2832 T_{CALC} \quad T(36) = T_{CALC} + T_{COM} \quad T_{CALC} = 48 \times (2 \text{ moltiplicazioni} + 3 \text{ somme}) = 16h + \\ T_{CALC} = 420 + 16h = 86h \quad T_{CALC}$$

$$T_{COM} = 15 \times 48 = p-3 \times cost = 420 \quad T_{COM} \quad T(36) = 16h + 86h = 230h \quad S(36) = \frac{2832}{230h} = 1,23$$

$$\bar{C}(36) = \frac{1,23}{36} = 0,03 \quad \theta_{PC} = \frac{T_{COM}}{T_{CALC}} = \frac{86h}{86h} = 1,0 \quad \theta_{HT} = (36 \times 230h) - 2832 = 34032$$

Wore Totale operazioni = 2832

$$\alpha = \frac{15 \times 48}{2832} = \frac{86h}{2832} \quad 1-\alpha = \frac{(48 \times 3) \times 16 + 2 \times (48)}{2832} = \frac{2036 + 96}{2832} = \frac{2132}{2832} \\ S(36) = \frac{1}{\frac{420}{2832} + \frac{2132}{2832}} = \frac{1}{\frac{252}{2832}} = 3,32 \quad \bar{C}(36) = \frac{3,32}{36} = 0,09$$

### Strategy E

Definizione Classica

$$T_{COM} = 2T_{CALC}$$

$\log_2 36$  Passi di comunicazione

$$T(3) = 2832 T_{CALC} \quad T(36) = T_{CALC} + T_{COM} \quad T_{CALC} = 48 \times (2 \text{ moltiplicazioni} + 3 \text{ somme}) = 16h + \\ T_{CALC} = 192 + 16h = 336 \quad T_{CALC}$$

$$T_{COM} = 6 \times 48 = \log_2 16 \times cost = 192 \quad T_{COM} \quad T(36) = 336 + 192 = 528 \quad S(36) = \frac{2832}{528} = 5,33 \\ \bar{C}(36) = \frac{5,33}{36} = 0,14 \quad \theta_{PC} = \frac{T_{COM}}{T_{CALC}} = \frac{192}{336} = 0,56 \quad \theta_{HT} = (36 \times 528) - 2832 = 8688$$

Wore generalizzato

$$\alpha_1 = \frac{48}{2832} = \frac{1}{58}, \quad \alpha_2 = \frac{2 \times 48}{2832}, \quad \alpha_3 = \frac{48 \times 48}{2832}, \quad \alpha_4 = \frac{48 \times 48}{2832} \quad \alpha_5 = \frac{2036 + 96}{2832} \quad \alpha_{36} = \frac{48 \times 3 \times 16 + 2 \times 48}{2832} = \frac{2132}{2832} \\ S(36) = \frac{1}{\frac{1}{58} + \frac{2}{58} + \frac{1}{58} + \frac{8}{58} + \frac{2132}{2832}} = \frac{1}{\frac{4}{58} + \frac{2132}{2832}} = 8,76 \quad \bar{C}(36) = \frac{8,76}{36} = 0,24$$

Wore

$$\alpha = \frac{1}{58}, \quad 1-\alpha = \frac{2984}{2832} \quad S(36) = \frac{1}{\frac{1}{58} + \frac{2984}{2832}} = \frac{1}{\frac{30}{58}} = 12,75 \quad \bar{C}(36) = \frac{12,75}{36} = 0,35$$

$$T(p) = \left( 48 + \frac{192}{36} \right) + 336 + COM = (48 + 48) + 336 = 472 + T_{CALC}$$

### Strategy F - Il risultato deve andare al un processore

Ogni processo viene posizionato su una griglia  $4 \times 6$ . I primi 8 processori vengono assegnati  $A_{LOC} \in \mathbb{R}^2$  e  $b_{i,j} \in \mathbb{R}^2$  invece l'ultimo 8 processori ricevono  $A_{LOC} \in \mathbb{R}^{2 \times 2}$ ,  $b_{i,j} \in \mathbb{R}^2$ . Non accade

Ogni processo viene posizionato su una griglia  $4 \times h$ . I primi  $\frac{h}{2}$  processori vengono eseguiti  $A_{loc} \in \mathbb{R}^4$  e  $b_{loc} \in \mathbb{R}^8$ , invece gli ultimi  $\frac{h}{2}$  processori ricevono  $A_{loc} \in \mathbb{R}^{12 \times 8}$  e  $b_{loc} \in \mathbb{R}^8$ . Ogni processo calcola il prodotto tra  $A_{loc}$  e  $b_{loc}$ , quindi restituisce un vettore di dimensione  $x_{loc} \in \mathbb{R}^{12}$ . Il processo può dunque sommare il suo vettore con i vettori dei processi sulla stessa riga. Infine, po' dunque moltiplicare il risultato della somma per la dimensione griglia  $- \frac{h}{2} \times h$ .

### Definizione Classica

$$T(s) = 2832 T_{ACC} \quad T(16) = T_{ACC} + T_{COM} \quad t_{com} = 2T_{ACC} \quad T_{ACC} = \frac{32 \times (8 \text{ mult. di colonna})}{360} + 3 \text{ dim-3} \times 12 = 360 + 96 = 216 \text{ T}_{ACC} \quad T_{COM} = 3 \times 32 + 3 \times 32$$

somma

$$T_{COM} = 72 \text{ T}_{COM} \quad T(16) = 236 + 36h = 236 + T_{ACC} \quad S(16) = \frac{2832}{360} = 7,86 \quad E(16) = \frac{216}{360} = 0,6$$

$$Oht = 36 \times 360 - 2832 = 2928 \quad Ohc = \frac{T_{COM}}{T_{ACC}} = \frac{36h}{236} = \frac{2}{3} = 0,6$$

Non generalizzata e non viene eseguita tutto in parallelo  $T(P) = T(16)$

$$\alpha_3 = \alpha_{16} = \frac{32 \times (8+7) \times 8}{360} + 3 \times 32 \times (7+6) = 1600 + 1932 = 2532$$

$$P_0 - P_1 \quad P_2 - P_3 \quad J_h = 3 \times 12$$

$$P_4 - P_5 \quad P_6 - P_7$$

3. Si enuncino le definizioni di:
- Overhead comunicazione
  - Efficienza
  - Overhead Totale
  - Speed up, Speed up-ideale
- e si esprima la relazione tra di essi, riscrivendo la prima in funzione del secondo.

Per enunciare la definizione di efficienza, enunciamo prima lo Speed up che rappresenta il rapporto tra il tempo di esecuzione del programma in 1 solo processo ed il tempo di esecuzione del programma in  $P$  processori. Risulta quindi la relazione del Tempo di esecuzione in  $P$  processori sul tempo di esecuzione su 1 processo.

$$S(P) = \frac{T(s)}{T(P)} \quad \text{Definiamo Speed up-ideale, quando lo Speed up-ideale rapportato a } T(s) \text{ è uguale proprio a } P. \quad \text{E quindi vale a dire che } T(s) = P \times T(P) \text{ e quindi i performance parallelizzabili.}$$

L'Overhead Totale rappresenta quanto differisce lo speed up da quello ideale.

$$Oht(P) = (P \times T(P)) - T(s) \quad \text{Se } T(s) \text{ è proprio uguale a } P \times T(P) \text{ allora l'overhead Totale misura 0. Grazie a queste informazioni, possiamo accrescere l'efficienza. Essa misura il rapporto tra lo speed up ed il numero dei processori. } E(P) = \frac{S(P)}{P} \rightarrow \frac{T(s)}{P \times T(P)} \rightarrow \frac{T(s)}{Oht(P) + T(s)} \quad \text{Quindi possiamo misurare l'efficienza in termi di overhead}$$

L'overhead di comunicazione invece, rappresenta il rapporto tra il tempo impiegato nello scambio di informazioni tra i processi, quindi  $T_{COM}(P)$  ed il tempo impiegato a calcolare le informazioni, ovvero  $T_{ACC}$ .  $Ohc = \frac{T_{COM}}{T_{ACC}}$   
 Il tempo di esecuzione invece, rappresenta una somma  $T(s) = T_g + T_p$  che rappresentano rispettivamente, tempo di esecuzione in sequenziale e tempo di esecuzione in parallelo su  $P$  processori.  $T(P) = (T_g + T_p) \cdot ACC + T_{COM}$  Osservate mai si definisca che  $T_g = T_{ACC}$  poiché non c'è né comunicazione, mentre  $T_p = T_{ACC} + T_{COM}$  in parallelo, abbiamo, in più, di comunicazione di  $T_{COM}$  al calcolo.  $T(P) = T_g + (T_{ACC}(P) + T_{COM}(P)) \cdot ACC + T_{COM}(P)$

Si consideri l'API OpenMP.

- Si descrivano i costrutti di Work-Sharing.
- Si descrivano tre clausole che siano applicabili ognuna ad almeno un costrutto descritto al punto precedente.

Prima di parlare dei costrutti, facciamo un piccolo preambolo: Utilizzando l'API OpenMP andiamo a sfruttare le architetture SIMD-SIMD (Multiple Instruction Multiple Data - Shared Memory), le quali vengono divise in Thread di esecuzione ed ogni Thread di un processore, può accedere alla stessa memoria. Viene utilizzato lo schema FORK-JOIN per gestire il parallelismo fra i Thread, per aprire un blocco parallelo bisogna usare il comando #pragma omp parallel. Come ulteriori informazioni a questo comando si possono aggiungere, prima della lettura codice da eseguire in modo parallelo, alcuni costrutti, denominati costitutivi Work-Sharing e sono: Tra:

- 1) For: divide automaticamente le iterazioni del ciclo fra i Thread. Non si è scelto un while poiché non si sa quando termina il ciclo.
- 2) Single: forza l'esecuzione di un blocco ad un Thread in particolare
- 3) Sections: Divide manualmente i vari block di codice ed assegna l'esecuzione a singoli Thread. Una volta inseriti i costutivi è possibile definire alcune clausole le quali permettono di fornire informazioni aggiuntive al blocco del parallelismo:
  - 1) Private (list): Indica una lista di variabili che ogni Thread riserva in una memoria privata, queste variabili vengono inizializzate per ogni Thread.
  - 2) Shared (list): Indica una lista di variabili condivise che ogni Thread può accedere e modificare.
  - 3) Nowait: Rimuove la barriera implicita di sincronizzazione al termine del blocco parallelizzabile, quel commento terminato ogni Thread prosegue per le sue strade.

6. Si descrivano i tipi di parallelismo studiati, evidenziando le differenze e servendosi di esempi.

Secondo la Tassonomia di Flynn, sono stati studiati 4 tipi di Architetture di cui 3 sono parallelizzabili.

- 1) MIMD (Multiple Instruction Single Data) è una architettura che sfrutta il Parallelismo Temporale la quale lavora su architetture basate su pipeline, un esempio potrebbe essere l'architettura di un ALU, la quale riceve più istruzioni lavora su singoli dati alla volta.
- 2) SIMD (Single Instruction Multiple Data) è una architettura che sfrutta l'architettura della CPU avendo una singola CU (Control Unit) e più ALU (Arithmetic Logic Unit) le quali lavorano in parallelo. Questa architettura sfrutta sia il parallelismo temporale che il uso di pipeline, che il Parallelismo Spaziale la quale sfrutta singole istruzioni della CU in questo caso è gestire l'interazione multiplo dei dati con varie ALU.
- 3) MIMD / Multiple Instruction Multiple Data) è una architettura che sfrutta il Parallelismo Asincrono, ovvero quel Parallelismo completo che consente nel lavorare compiuta istruzione alla volta o più due contemporaneamente. Un esempio possono trovarsi inclusi al multi processor.

45x46 18

1. Si consideri il prodotto matrice per vettore  $A \cdot b = c$ , con  $A \in \mathbb{R}^{n \times n}$ ,  $b \in \mathbb{R}^n$  su un'architettura MIMD-DM, costituita da 4 processori. La matrice A è distribuita per blocchi di colonne.

- Che dimensione hanno i blocchi  $A_{i,:}$  e come deve essere distribuito il vettore  $b$ ?
- Si descriva l'algoritmo relativo alla realizzazione del prodotto, affinché ogni processore abbia l'intero vettore risultato  $c$ .
- Si calcolino speed-up ed efficienza, secondo la legge di Ware-Amdhal generalizzata, definendo, where

$$1. \text{ a Processori} \quad 2. \text{ 1 m.} \quad 3. \text{ 1. m.} \quad 4. \text{ 1. m.} \quad 5. \text{ 1. m.} \quad 6. \text{ 1. m.} \quad 7. \text{ 1. m.} \quad 8. \text{ 1. m.} \quad 9. \text{ 1. m.} \quad 10. \text{ 1. m.} \quad 11. \text{ 1. m.} \quad 12. \text{ 1. m.} \quad 13. \text{ 1. m.} \quad 14. \text{ 1. m.} \quad 15. \text{ 1. m.} \quad 16. \text{ 1. m.} \quad 17. \text{ 1. m.} \quad 18. \text{ 1. m.} \quad 19. \text{ 1. m.} \quad 20. \text{ 1. m.} \quad 21. \text{ 1. m.} \quad 22. \text{ 1. m.} \quad 23. \text{ 1. m.} \quad 24. \text{ 1. m.} \quad 25. \text{ 1. m.} \quad 26. \text{ 1. m.} \quad 27. \text{ 1. m.} \quad 28. \text{ 1. m.} \quad 29. \text{ 1. m.} \quad 30. \text{ 1. m.} \quad 31. \text{ 1. m.} \quad 32. \text{ 1. m.} \quad 33. \text{ 1. m.} \quad 34. \text{ 1. m.} \quad 35. \text{ 1. m.} \quad 36. \text{ 1. m.} \quad 37. \text{ 1. m.} \quad 38. \text{ 1. m.} \quad 39. \text{ 1. m.} \quad 40. \text{ 1. m.} \quad 41. \text{ 1. m.} \quad 42. \text{ 1. m.} \quad 43. \text{ 1. m.} \quad 44. \text{ 1. m.} \quad 45. \text{ 1. m.} \quad 46. \text{ 1. m.} \quad 47. \text{ 1. m.} \quad 48. \text{ 1. m.} \quad 49. \text{ 1. m.} \quad 50. \text{ 1. m.} \quad 51. \text{ 1. m.} \quad 52. \text{ 1. m.} \quad 53. \text{ 1. m.} \quad 54. \text{ 1. m.} \quad 55. \text{ 1. m.} \quad 56. \text{ 1. m.} \quad 57. \text{ 1. m.} \quad 58. \text{ 1. m.} \quad 59. \text{ 1. m.} \quad 60. \text{ 1. m.} \quad 61. \text{ 1. m.} \quad 62. \text{ 1. m.} \quad 63. \text{ 1. m.} \quad 64. \text{ 1. m.} \quad 65. \text{ 1. m.} \quad 66. \text{ 1. m.} \quad 67. \text{ 1. m.} \quad 68. \text{ 1. m.} \quad 69. \text{ 1. m.} \quad 70. \text{ 1. m.} \quad 71. \text{ 1. m.} \quad 72. \text{ 1. m.} \quad 73. \text{ 1. m.} \quad 74. \text{ 1. m.} \quad 75. \text{ 1. m.} \quad 76. \text{ 1. m.} \quad 77. \text{ 1. m.} \quad 78. \text{ 1. m.} \quad 79. \text{ 1. m.} \quad 80. \text{ 1. m.} \quad 81. \text{ 1. m.} \quad 82. \text{ 1. m.} \quad 83. \text{ 1. m.} \quad 84. \text{ 1. m.} \quad 85. \text{ 1. m.} \quad 86. \text{ 1. m.} \quad 87. \text{ 1. m.} \quad 88. \text{ 1. m.} \quad 89. \text{ 1. m.} \quad 90. \text{ 1. m.} \quad 91. \text{ 1. m.} \quad 92. \text{ 1. m.} \quad 93. \text{ 1. m.} \quad 94. \text{ 1. m.} \quad 95. \text{ 1. m.} \quad 96. \text{ 1. m.} \quad 97. \text{ 1. m.} \quad 98. \text{ 1. m.} \quad 99. \text{ 1. m.} \quad 100. \text{ 1. m.} \quad 101. \text{ 1. m.} \quad 102. \text{ 1. m.} \quad 103. \text{ 1. m.} \quad 104. \text{ 1. m.} \quad 105. \text{ 1. m.} \quad 106. \text{ 1. m.} \quad 107. \text{ 1. m.} \quad 108. \text{ 1. m.} \quad 109. \text{ 1. m.} \quad 110. \text{ 1. m.} \quad 111. \text{ 1. m.} \quad 112. \text{ 1. m.} \quad 113. \text{ 1. m.} \quad 114. \text{ 1. m.} \quad 115. \text{ 1. m.} \quad 116. \text{ 1. m.} \quad 117. \text{ 1. m.} \quad 118. \text{ 1. m.} \quad 119. \text{ 1. m.} \quad 120. \text{ 1. m.} \quad 121. \text{ 1. m.} \quad 122. \text{ 1. m.} \quad 123. \text{ 1. m.} \quad 124. \text{ 1. m.} \quad 125. \text{ 1. m.} \quad 126. \text{ 1. m.} \quad 127. \text{ 1. m.} \quad 128. \text{ 1. m.} \quad 129. \text{ 1. m.} \quad 130. \text{ 1. m.} \quad 131. \text{ 1. m.} \quad 132. \text{ 1. m.} \quad 133. \text{ 1. m.} \quad 134. \text{ 1. m.} \quad 135. \text{ 1. m.} \quad 136. \text{ 1. m.} \quad 137. \text{ 1. m.} \quad 138. \text{ 1. m.} \quad 139. \text{ 1. m.} \quad 140. \text{ 1. m.} \quad 141. \text{ 1. m.} \quad 142. \text{ 1. m.} \quad 143. \text{ 1. m.} \quad 144. \text{ 1. m.} \quad 145. \text{ 1. m.} \quad 146. \text{ 1. m.} \quad 147. \text{ 1. m.} \quad 148. \text{ 1. m.} \quad 149. \text{ 1. m.} \quad 150. \text{ 1. m.} \quad 151. \text{ 1. m.} \quad 152. \text{ 1. m.} \quad 153. \text{ 1. m.} \quad 154. \text{ 1. m.} \quad 155. \text{ 1. m.} \quad 156. \text{ 1. m.} \quad 157. \text{ 1. m.} \quad 158. \text{ 1. m.} \quad 159. \text{ 1. m.} \quad 160. \text{ 1. m.} \quad 161. \text{ 1. m.} \quad 162. \text{ 1. m.} \quad 163. \text{ 1. m.} \quad 164. \text{ 1. m.} \quad 165. \text{ 1. m.} \quad 166. \text{ 1. m.} \quad 167. \text{ 1. m.} \quad 168. \text{ 1. m.} \quad 169. \text{ 1. m.} \quad 170. \text{ 1. m.} \quad 171. \text{ 1. m.} \quad 172. \text{ 1. m.} \quad 173. \text{ 1. m.} \quad 174. \text{ 1. m.} \quad 175. \text{ 1. m.} \quad 176. \text{ 1. m.} \quad 177. \text{ 1. m.} \quad 178. \text{ 1. m.} \quad 179. \text{ 1. m.} \quad 180. \text{ 1. m.} \quad 181. \text{ 1. m.} \quad 182. \text{ 1. m.} \quad 183. \text{ 1. m.} \quad 184. \text{ 1. m.} \quad 185. \text{ 1. m.} \quad 186. \text{ 1. m.} \quad 187. \text{ 1. m.} \quad 188. \text{ 1. m.} \quad 189. \text{ 1. m.} \quad 190. \text{ 1. m.} \quad 191. \text{ 1. m.} \quad 192. \text{ 1. m.} \quad 193. \text{ 1. m.} \quad 194. \text{ 1. m.} \quad 195. \text{ 1. m.} \quad 196. \text{ 1. m.} \quad 197. \text{ 1. m.} \quad 198. \text{ 1. m.} \quad 199. \text{ 1. m.} \quad 200. \text{ 1. m.} \quad 201. \text{ 1. m.} \quad 202. \text{ 1. m.} \quad 203. \text{ 1. m.} \quad 204. \text{ 1. m.} \quad 205. \text{ 1. m.} \quad 206. \text{ 1. m.} \quad 207. \text{ 1. m.} \quad 208. \text{ 1. m.} \quad 209. \text{ 1. m.} \quad 210. \text{ 1. m.} \quad 211. \text{ 1. m.} \quad 212. \text{ 1. m.} \quad 213. \text{ 1. m.} \quad 214. \text{ 1. m.} \quad 215. \text{ 1. m.} \quad 216. \text{ 1. m.} \quad 217. \text{ 1. m.} \quad 218. \text{ 1. m.} \quad 219. \text{ 1. m.} \quad 220. \text{ 1. m.} \quad 221. \text{ 1. m.} \quad 222. \text{ 1. m.} \quad 223. \text{ 1. m.} \quad 224. \text{ 1. m.} \quad 225. \text{ 1. m.} \quad 226. \text{ 1. m.} \quad 227. \text{ 1. m.} \quad 228. \text{ 1. m.} \quad 229. \text{ 1. m.} \quad 230. \text{ 1. m.} \quad 231. \text{ 1. m.} \quad 232. \text{ 1. m.} \quad 233. \text{ 1. m.} \quad 234. \text{ 1. m.} \quad 235. \text{ 1. m.} \quad 236. \text{ 1. m.} \quad 237. \text{ 1. m.} \quad 238. \text{ 1. m.} \quad 239. \text{ 1. m.} \quad 240. \text{ 1. m.} \quad 241. \text{ 1. m.} \quad 242. \text{ 1. m.} \quad 243. \text{ 1. m.} \quad 244. \text{ 1. m.} \quad 245. \text{ 1. m.} \quad 246. \text{ 1. m.} \quad 247. \text{ 1. m.} \quad 248. \text{ 1. m.} \quad 249. \text{ 1. m.} \quad 250. \text{ 1. m.} \quad 251. \text{ 1. m.} \quad 252. \text{ 1. m.} \quad 253. \text{ 1. m.} \quad 254. \text{ 1. m.} \quad 255. \text{ 1. m.} \quad 256. \text{ 1. m.} \quad 257. \text{ 1. m.} \quad 258. \text{ 1. m.} \quad 259. \text{ 1. m.} \quad 260. \text{ 1. m.} \quad 261. \text{ 1. m.} \quad 262. \text{ 1. m.} \quad 263. \text{ 1. m.} \quad 264. \text{ 1. m.} \quad 265. \text{ 1. m.} \quad 266. \text{ 1. m.} \quad 267. \text{ 1. m.} \quad 268. \text{ 1. m.} \quad 269. \text{ 1. m.} \quad 270. \text{ 1. m.} \quad 271. \text{ 1. m.} \quad 272. \text{ 1. m.} \quad 273. \text{ 1. m.} \quad 274. \text{ 1. m.} \quad 275. \text{ 1. m.} \quad 276. \text{ 1. m.} \quad 277. \text{ 1. m.} \quad 278. \text{ 1. m.} \quad 279. \text{ 1. m.} \quad 280. \text{ 1. m.} \quad 281. \text{ 1. m.} \quad 282. \text{ 1. m.} \quad 283. \text{ 1. m.} \quad 284. \text{ 1. m.} \quad 285. \text{ 1. m.} \quad 286. \text{ 1. m.} \quad 287. \text{ 1. m.} \quad 288. \text{ 1. m.} \quad 289. \text{ 1. m.} \quad 290. \text{ 1. m.} \quad 291. \text{ 1. m.} \quad 292. \text{ 1. m.} \quad 293. \text{ 1. m.} \quad 294. \text{ 1. m.} \quad 295. \text{ 1. m.} \quad 296. \text{ 1. m.} \quad 297. \text{ 1. m.} \quad 298. \text{ 1. m.} \quad 299. \text{ 1. m.} \quad 300. \text{ 1. m.} \quad 301. \text{ 1. m.} \quad 302. \text{ 1. m.} \quad 303. \text{ 1. m.} \quad 304. \text{ 1. m.} \quad 305. \text{ 1. m.} \quad 306. \text{ 1. m.} \quad 307. \text{ 1. m.} \quad 308. \text{ 1. m.} \quad 309. \text{ 1. m.} \quad 310. \text{ 1. m.} \quad 311. \text{ 1. m.} \quad 312. \text{ 1. m.} \quad 313. \text{ 1. m.} \quad 314. \text{ 1. m.} \quad 315. \text{ 1. m.} \quad 316. \text{ 1. m.} \quad 317. \text{ 1. m.} \quad 318. \text{ 1. m.} \quad 319. \text{ 1. m.} \quad 320. \text{ 1. m.} \quad 321. \text{ 1. m.} \quad 322. \text{ 1. m.} \quad 323. \text{ 1. m.} \quad 324. \text{ 1. m.} \quad 325. \text{ 1. m.} \quad 326. \text{ 1. m.} \quad 327. \text{ 1. m.} \quad 328. \text{ 1. m.} \quad 329. \text{ 1. m.} \quad 330. \text{ 1. m.} \quad 331. \text{ 1. m.} \quad 332. \text{ 1. m.} \quad 333. \text{ 1. m.} \quad 334. \text{ 1. m.} \quad 335. \text{ 1. m.} \quad 336. \text{ 1. m.} \quad 337. \text{ 1. m.} \quad 338. \text{ 1. m.} \quad 339. \text{ 1. m.} \quad 340. \text{ 1. m.} \quad 341. \text{ 1. m.} \quad 342. \text{ 1. m.} \quad 343. \text{ 1. m.} \quad 344. \text{ 1. m.} \quad 345. \text{ 1. m.} \quad 346. \text{ 1. m.} \quad 347. \text{ 1. m.} \quad 348. \text{ 1. m.} \quad 349. \text{ 1. m.} \quad 350. \text{ 1. m.} \quad 351. \text{ 1. m.} \quad 352. \text{ 1. m.} \quad 353. \text{ 1. m.} \quad 354. \text{ 1. m.} \quad 355. \text{ 1. m.} \quad 356. \text{ 1. m.} \quad 357. \text{ 1. m.} \quad 358. \text{ 1. m.} \quad 359. \text{ 1. m.} \quad 360. \text{ 1. m.} \quad 361. \text{ 1. m.} \quad 362. \text{ 1. m.} \quad 363. \text{ 1. m.} \quad 364. \text{ 1. m.} \quad 365. \text{ 1. m.} \quad 366. \text{ 1. m.} \quad 367. \text{ 1. m.} \quad 368. \text{ 1. m.} \quad 369. \text{ 1. m.} \quad 370. \text{ 1. m.} \quad 371. \text{ 1. m.} \quad 372. \text{ 1. m.} \quad 373. \text{ 1. m.} \quad 374. \text{ 1. m.} \quad 375. \text{ 1. m.} \quad 376. \text{ 1. m.} \quad 377. \text{ 1. m.} \quad 378. \text{ 1. m.} \quad 379. \text{ 1. m.} \quad 380. \text{ 1. m.} \quad 381. \text{ 1. m.} \quad 382. \text{ 1. m.} \quad 383. \text{ 1. m.} \quad 384. \text{ 1. m.} \quad 385. \text{ 1. m.} \quad 386. \text{ 1. m.} \quad 387. \text{ 1. m.} \quad 388. \text{ 1. m.} \quad 389. \text{ 1. m.} \quad 390. \text{ 1. m.} \quad 391. \text{ 1. m.} \quad 392. \text{ 1. m.} \quad 393. \text{ 1. m.} \quad 394. \text{ 1. m.} \quad 395. \text{ 1. m.} \quad 396. \text{ 1. m.} \quad 397. \text{ 1. m.} \quad 398. \text{ 1. m.} \quad 399. \text{ 1. m.} \quad 400. \text{ 1. m.} \quad 401. \text{ 1. m.} \quad 402. \text{ 1. m.} \quad 403. \text{ 1. m.} \quad 404. \text{ 1. m.} \quad 405. \text{ 1. m.} \quad 406. \text{ 1. m.} \quad 407. \text{ 1. m.} \quad 408. \text{ 1. m.} \quad 409. \text{ 1. m.} \quad 410. \text{ 1. m.} \quad 411. \text{ 1. m.} \quad 412. \text{ 1. m.} \quad 413. \text{ 1. m.} \quad 414. \text{ 1. m.} \quad 415. \text{ 1. m.} \quad 416. \text{ 1. m.} \quad 417. \text{ 1. m.} \quad 418. \text{ 1. m.} \quad 419. \text{ 1. m.} \quad 420. \text{ 1. m.} \quad 421. \text{ 1. m.} \quad 422. \text{ 1. m.} \quad 423. \text{ 1. m.} \quad 424. \text{ 1. m.} \quad 425. \text{ 1. m.} \quad 426. \text{ 1. m.} \quad 427. \text{ 1. m.} \quad 428. \text{ 1. m.} \quad 429. \text{ 1. m.} \quad 430. \text{ 1. m.} \quad 431. \text{ 1. m.} \quad 432. \text{ 1. m.} \quad 433. \text{ 1. m.} \quad 434. \text{ 1. m.} \quad 435. \text{ 1. m.} \quad 436. \text{ 1. m.} \quad 437. \text{ 1. m.} \quad 438. \text{ 1. m.} \quad 439. \text{ 1. m.} \quad 440. \text{ 1. m.} \quad 441. \text{ 1. m.} \quad 442. \text{ 1. m.} \quad 443. \text{ 1. m.} \quad 444. \text{ 1. m.} \quad 445. \text{ 1. m.} \quad 446. \text{ 1. m.} \quad 447. \text{ 1. m.} \quad 448. \text{ 1. m.} \quad 449. \text{ 1. m.} \quad 450. \text{ 1. m.} \quad 451. \text{ 1. m.} \quad 452. \text{ 1. m.} \quad 453. \text{ 1. m.} \quad 454. \text{ 1. m.} \quad 455. \text{ 1. m.} \quad 456. \text{ 1. m.} \quad 457. \text{ 1. m.} \quad 458. \text{ 1. m.} \quad 459. \text{ 1. m.} \quad 460. \text{ 1. m.} \quad 461. \text{ 1. m.} \quad 462. \text{ 1. m.} \quad 463. \text{ 1. m.} \quad 464. \text{ 1. m.} \quad 465. \text{ 1. m.} \quad 466. \text{ 1. m.} \quad 467. \text{ 1. m.} \quad 468. \text{ 1. m.} \quad 469. \text{ 1. m.} \quad 470. \text{ 1. m.} \quad 471. \text{ 1. m.} \quad 472. \text{ 1. m.} \quad 473. \text{ 1. m.} \quad 474. \text{ 1. m.} \quad 475. \text{ 1. m.} \quad 476. \text{ 1. m.} \quad 477. \text{ 1. m.} \quad 478. \text{ 1. m.} \quad 479. \text{ 1. m.} \quad 480. \text{ 1. m.} \quad 481. \text{ 1. m.} \quad 482. \text{ 1. m.} \quad 483. \text{ 1. m.} \quad 484. \text{ 1. m.} \quad 485. \text{ 1. m.} \quad 486. \text{ 1. m.} \quad 487. \text{ 1. m.} \quad 488. \text{ 1. m.} \quad 489. \text{ 1. m.} \quad 490. \text{ 1. m.} \quad 491. \text{ 1. m.} \quad 492. \text{ 1. m.} \quad 493. \text{ 1. m.} \quad 494. \text{ 1. m.} \quad 495. \text{ 1. m.} \quad 496. \text{ 1. m.} \quad 497. \text{ 1. m.} \quad 498. \text{ 1. m.} \quad 499. \text{ 1. m.} \quad 500. \text{ 1. m.} \quad 501. \text{ 1. m.} \quad 502. \text{ 1. m.} \quad 503. \text{ 1. m.} \quad 504. \text{ 1. m.} \quad 505. \text{ 1. m.} \quad 506. \text{ 1. m.} \quad 507. \text{ 1. m.} \quad 508. \text{ 1. m.} \quad 509. \text{ 1. m.} \quad 510. \text{ 1. m.} \quad 511. \text{ 1. m.} \quad 512. \text{ 1. m.} \quad 513. \text{ 1. m.} \quad 514. \text{ 1. m.} \quad 515. \text{ 1. m.} \quad 516. \text{ 1. m.} \quad 517. \text{ 1. m.} \quad 518. \text{ 1. m.} \quad 519. \text{ 1. m.} \quad 520. \text{ 1. m.} \quad 521. \text{ 1. m.} \quad 522. \text{ 1. m.} \quad 523. \text{ 1. m.} \quad 524. \text{ 1. m.} \quad 525. \text{ 1. m.} \quad 526. \text{ 1. m.} \quad 527. \text{ 1. m.} \quad 528. \text{ 1. m.} \quad 529. \text{ 1. m.} \quad 530. \text{ 1. m.} \quad 531. \text{ 1. m.} \quad 532. \text{ 1. m.} \quad 533. \text{ 1. m.} \quad 534. \text{ 1. m.} \quad 535. \text{ 1. m.} \quad 536. \text{ 1. m.} \quad 537. \text{ 1. m.} \quad 538. \text{ 1. m.} \quad 539. \text{ 1. m.} \quad 540. \text{ 1. m.} \quad 541. \text{ 1. m.} \quad 542. \text{ 1. m.} \quad 543. \text{ 1. m.} \quad 544. \text{ 1. m.} \quad 545. \text{ 1. m.} \quad 546. \text{ 1. m.} \quad 547. \text{ 1. m.} \quad 548. \text{ 1. m.} \quad 549. \text{ 1. m.} \quad 550. \text{ 1. m.} \quad 551. \text{ 1. m.} \quad 552. \text{ 1. m.} \quad 553. \text{ 1. m.} \quad 554. \text{ 1. m.} \quad 555. \text{ 1. m.} \quad 556. \text{ 1. m.} \quad 557. \text{ 1. m.} \quad 558. \text{ 1. m.} \quad 559. \text{ 1. m.} \quad 560. \text{ 1. m.} \quad 561. \text{ 1. m.} \quad 562. \text{ 1. m.} \quad 563. \text{ 1. m.} \quad 564. \text{ 1. m.} \quad 565. \text{ 1. m.} \quad 566. \text{ 1. m.} \quad 567. \text{ 1. m.} \quad 568. \text{ 1. m.} \quad 569. \text{ 1. m.} \quad 570. \text{ 1. m.} \quad 571. \text{ 1. m.} \quad 572. \text{ 1. m.} \quad 573. \text{ 1. m.} \quad 574. \text{ 1. m.} \quad 575. \text{ 1. m.} \quad 576. \text{ 1. m.} \quad 577. \text{ 1. m.} \quad 578. \text{ 1. m.} \quad 579. \text{ 1. m.} \quad 580. \text{ 1. m.} \quad 581. \text{ 1. m.} \quad 582. \text{ 1. m.} \quad 583. \text{ 1. m.} \quad 584. \text{ 1. m.} \quad 585. \text{ 1. m.} \quad 586. \text{ 1. m.} \quad 587. \text{ 1. m.} \quad 588. \text{ 1. m.} \quad 589. \text{ 1. m.} \quad 590. \text{ 1. m.} \quad 591. \text{ 1. m.} \quad 592. \text{ 1. m.} \quad 593. \text{ 1. m.} \quad 594. \text{ 1. m.} \quad 595. \text{ 1. m.} \quad 596. \text{ 1. m.} \quad 597. \text{ 1. m.} \quad 598. \text{ 1. m.} \quad 599. \text{ 1. m.} \quad 600. \text{ 1. m.} \quad 601. \text{ 1. m.} \quad 602. \text{ 1. m.} \quad 603. \text{ 1. m.} \quad 604. \text{ 1. m.} \quad 605. \text{ 1. m.} \quad 606. \text{ 1. m.} \quad 607. \text{ 1. m.} \quad 608. \text{ 1. m.} \quad 609. \text{ 1. m.} \quad 610. \text{ 1. m.} \quad 611. \text{ 1. m.} \quad 612. \text{ 1. m.} \quad 613. \text{ 1. m.} \quad 614. \text{ 1. m.} \quad 615. \text{ 1. m.} \quad 616. \text{ 1. m.} \quad 617. \text{$$

l'intero vettore risultato c.

c. Si calcolino speed-up ed efficienza, secondo la legge di Ware-Amdahl generalizzata, defin. classica, nere

a) Ogni processore riceverà una parte della matrice A ed una parte del vettore b. Visto che la matrice A è distribuita per blocchi di colonne, se non è divisibile per 4, pertanto 2 processori riceveranno  $A_{loc} \in \mathbb{R}^{45 \times 4}$  e  $b_{loc} \in \mathbb{R}^4$ , mentre gli ultimi 2 processori riceveranno  $A_{loc} \in \mathbb{R}^{45 \times 5}$  e  $b_{loc} \in \mathbb{R}^5$ . Il vettore b viene distribuito tra i 4 processori.

b) Ogni processore calcolerà il prodotto tra il proprio  $A_{loc}$  ed il proprio  $b_{loc}$ , il cui risultato  $A_{loc} \times b_{loc} = c_{loc}$  quindi ogni processore invia allo fine un vettore  $c_{loc}$  il quale deve essere sommato a tutti i risultati degli altri processori affinché si ottenga l'insieme finale ovvero il vettore c. Quindi si tratta di calcolare una somma fra i processori sì, possono quindi effettuare 3 strategie, poiché noi vogliamo che il risultato venga calcolato a tutti i processori, sceglieremo la terza strategia. Vogliamo mesi ed ogni passo di comunicazione, per un totale di  $\log_2 P$  passi, compreso il processo. Se quel colpo ogni passo avviene al più tardi il doppio del passo precedente, pertanto da un minimo di passi 3 al massimo di passi 4. Vogliamo eseguire 2 passi di comunicazione, al passo uno ogni processore invia e riceve dal/da processore il vettore preso 2 nella coppia. I due sommano il proprio  $c_{loc}$  con quello che viene inviato dall'altro processore. Al passo 2 eseguono quindi  $6 \times 65$  somme. Infine al passo 3 ogni processore invia il risultato finale.

### c) Ware-Amdahl generalizzato

Per calcolare lo speedup ed efficienza, la legge di Ware consente di calcolare  $T(s) = T_S + T_P$  e

$$T(P) = T_S + \frac{T_P}{P} \quad \text{ponendo quindi } T_S = \alpha \quad \text{e} \quad T_P = 1 - \alpha \rightarrow T(s) = 1 \quad T(P) = \alpha + \left( \frac{1-\alpha}{P} \right)$$

$S(P) = \frac{1}{\alpha + \left( \frac{1-\alpha}{P} \right)}$  Soltanamente che la legge di Ware-Amdahl generalizzata permette di avere più  $\alpha$ , considerate come  $\alpha_k$  con  $k$ -processori, equivalenti alle operazioni eseguite in parallelo da k processori. Considerando questa strategia tutto è eseguito in funzione di k processori, ma se ci sono operazioni eseguite in parallelo da meno processori allora la legge di Ware generalizzata prenderebbe in considerazione.

$$\alpha_1 = 0 \quad \text{poiché tutto è eseguito in parallelo.} \quad \alpha_2 = 0 \quad \alpha_n = \frac{630}{2 \times 45 (45 \text{ moltiplicazioni} + 33 \text{ somme}) + 2 \times 45 (5 \text{ moltiplicazioni} + 450 \text{ somme}) + 8 \times 45 \times 4 \times 45 = 3800}$$

$$S(n) = \frac{1}{0 + \frac{3800}{3800}} = \frac{1}{0} = 4 \quad E(n) = \frac{S(n)}{n} = 1$$

$$\begin{aligned} & \alpha_1 = 2 \times 45 (45 \text{ moltiplicazioni} + 33 \text{ somme}) + \\ & 2 \times 45 (5 \text{ moltiplicazioni} + 450 \text{ somme}) + \\ & 8 \times 45 \times 4 \times 45 = 3800 \end{aligned}$$

### Definizione classica

$$T(s) = 45 \times (18 \text{ moltiplicazioni} + 17 \text{ somme}) = 1575 \quad T_{Acc} = T_{Acc} + T_{Com} \quad T_{Com} = 2T_{Acc}$$

$$T_{Acc} = 45 (5 \text{ moltiplicazioni} + 6 \text{ somme}) + 2 \times 45 = 495 \quad T_{Com} = 2 \times 45 = 90 \quad T(h) = 495 + 90 = 585 \quad T(h) = 585 \text{ sec}$$

$$S(n) = \frac{T(1)}{T(n)} = \frac{1575}{585} = \frac{5}{3} = 2,33 \quad E(n) = \frac{S(n)}{n} = 0,58 \quad \theta_{hc} = \frac{T_{Com}}{T_{Acc}} = \frac{90}{495} = \frac{4}{19} = 0,36$$

$$\theta_{ht} = 4 \times 675 - 1575 = 1325 \quad T(P) = T_S + \frac{T_P}{P} + T_{Com} = 0 + \frac{675}{4} + 90 = 348,75$$

2. Si consideri il prodotto Matrice-~~vettore~~, da eseguire su un'architettura MIMD a memoria distribuita. Si descrivano due possibili strategie di approccio alla soluzione del problema, tenendo conto della distribuzione dei dati e della ricompilazione del risultato finale, evidenziando le differenze e le analogie tra i due approcci scelti.

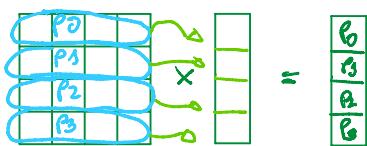
Prima di procedere con la discussione delle strategie, bisogna distribuire i dati prima  $T_{acc}$  i processori. Ogni processore riceverà un blocco di dati della matrice ed un blocco di dati del vettore. Poniamo la matrice di dimensione  $N \times N$ , il vettore di dim.

### Strategia 1

Nella Strategia 1 la matrice viene divisa in blocchi di righe ed il vettore viene inviato completamente ad ogni processo. Avendo quindi  $P$  processori che lavorano concorrentemente, la matrice  $A_{loc}$  che ciascun processo avrà di dimensione  $R \times M$  con  $R$  uguale a  $\frac{N}{P}$  se  $M \leq P = 0$  altrimenti  $\frac{N}{P} + 1$ . Il vettore  $b_{loc}$  sarà proprio grande abbastanza di dimensione  $M$ .

Ogni processo calcola una  $c_{loc}$  di dimensione  $R$ . Poiché ogni processo ha calcolato solo una parte del vettore completo, si risulta che ogni processo riceverà il risultato altrui tutti i processi si scambieranno il proprio vettore. Per un Totale di  $P \times (P-1) \times R$  operazioni. Se invece il risultato lo deve fornire un solo processo allora verremo eseguiti  $P-1 \times R$  comunicazioni.

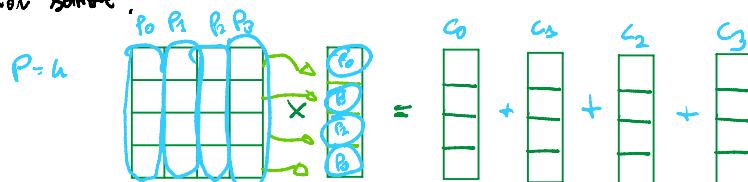
Ogni processo eseguirà quindi almeno



$$R \times (M \text{ moltiplicazioni} + M-1 \text{ somme})$$

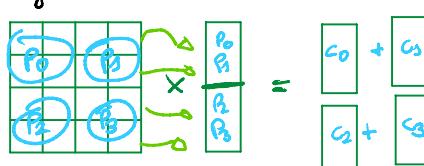
### Strategia 2

La Matrice in questione viene divisa in blocchi di colonne, ed il vettore viene distribuito tra i vari processi. Si considerano  $P$  processi che lavorano concorrentemente,  $A \in \mathbb{R}^{N \times M}$  e  $b \in \mathbb{R}^M$ . Ogni processo riceverà  $A_{loc} \in \mathbb{R}^{N \times R}$  e  $b_{loc} \in \mathbb{R}^R$  con  $R = \frac{M}{P}$  se  $M \leq P = 0$  altrimenti  $R = \frac{M}{P} + 1$ . Ogni processo calcolerà il prodotto fra  $A_{loc} \cdot b_{loc}$  il quale restituirà  $c_{loc} \in \mathbb{R}^N$ , ovvero una parte del vettore finale  $c$ . Ogni processo dovrà quindi sommare tutte le  $c_{loc}$  di tutti gli altri processi. Qui, si possono attuare 3 strategie, ovvero 3 strategie della somma, le quali permettono di far ottenere il risultato in uno solo oppure a tutti i processi. Se sceglieremo ad esempio la seconda strategia verremo a seguire 3 passi di comunicazione elettronico: i processi lavoreranno in coppia per sommare l'intero vettore fino ad arrivare alla somma finale che contiene 1 solo processo. Ogni processo quindi eseguirà almeno  $N \times (R \text{ moltiplicazioni} + R-1 \text{ somme})$ , senza tener conto delle ulteriori somme.



### Strategia 3

Considerando sempre la matrice  $A$  di dimensione  $A \in \mathbb{R}^{N \times N}$ , vettore  $b \in \mathbb{R}^N$  e  $P$  processi, questa volta i processi vedono disposti su una griglia quadrata, di dimensione  $\sqrt{P} \times \sqrt{P}$ . La matrice  $A$  viene distribuita per blocchi quadrati, a cui sono associati dei blocchi di righe e blocchi di colonne, mentre il vettore  $b$  viene distribuito in  $b_{loc} \in \mathbb{R}^{\frac{N}{\sqrt{P}}}$ . La matrice  $A$  viene distribuita in  $A_{loc} \in \mathbb{R}^{\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}}$ . Ogni processo calcolerà quindi  $c_{loc} \in \mathbb{R}^{\frac{N}{\sqrt{P}}}$  il quale avremo ancora da calcolare svolgendo  $\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} \text{ moltiplicazioni} + \frac{N}{\sqrt{P}} - 1 \text{ somme}$ . Una volta che tutti i processi hanno calcolato il proprio  $c_{loc}$ , i processi dovranno sommarsi con tutti gli altri processi presenti sulla stessa riga o colonna il proprio  $c_{loc}$  con il loro. Vediamo eseguire  $\sqrt{P} \times \sqrt{P} \times (\sqrt{P}-1 \times \frac{N}{\sqrt{P}})$  operazioni. Per tutte le righe e colonne. Se invece si vuole far sì che il risultato di un solo processo bisogna togliere le colonne. Infine ogni processo contiene un sottoinsieme del vettore finale e quindi ogni processo dovrà scambiarsi con gli altri processi sulla stessa colonna della griglia il proprio risultato. Per eseguire un Totale di  $\sqrt{P} \times \sqrt{P} \times \frac{N}{\sqrt{P}}$  comunicazioni,



2. Si consideri il prodotto Matrice-Matrice, da eseguire su un'architettura MIMD a memoria distribuita. Si descrivano due possibili strategie di approccio alla soluzione del problema, tenendo conto della distribuzione dei dati e della ricompilazione del risultato finale, evidenziando le differenze e le analogie tra i due approcci scelti.

Per TUTTE e TUTTE le strategie verremo denominare le 2 matrici:  $A \in \mathbb{R}^{M \times N}$  e  $B \in \mathbb{R}^{N \times L}$ ,  $P$  rappresenta il numero di processori.

### Strategia 1

Le due matrici vengono distribuite in: matrice  $A$  in blocchi di righe, mentre matrice  $B$  in blocchi di colonne.

$$R = \frac{N}{P} \text{ se } N \% P = 0 \quad \frac{N}{P} + s \text{ altrimenti} \quad Q = \frac{L}{P} \text{ se } L \% P = 0 \quad \frac{L}{P} + s \text{ altrimenti.} \quad \text{Ogni processo riceverà quindi}$$

$A_{loc} \in \mathbb{R}^{R \times \frac{N}{P}}$  mentre  $B_{loc} \in \mathbb{R}^{M \times Q}$ . Ogni processo potrà calcolare una matrice  $C_{loc} \in \mathbb{R}^{R \times Q}$  eseguendo

$R \times Q \times (M \text{ moltiplicazioni} + M - 3 \text{ somme})$ . Ogni processo dovrà ricevere i Bloc di tutti gli altri processori per un totale di  $p-1$  Bloc a processo, e calcolare altre  $p-1$  prodotti. Tra il proprio  $A_{loc}$  e TUTTI i Bloc di Tutti i processori. Eseguendo quindi un totale  $P \times R \times Q \times (R + L - 1)$  operazioni. Nonostante tutto, ogni processo contiene solo una sotto-matrica della matrice C finale quindi Tutti i processi dovranno scambiarsi la propria matrice  $C_{loc}$  di dimensione  $R \times L$  con p processo. Invia se si vuole che 1 solo processo debba ricevere il risultato finale, Tutti i processi convergono ad 1 solo processo.

### Strategia 2

Le due matrici vengono distribuite diversamente ovvero: Matrice  $A$  viene divisa in blocchi di colonne, mentre la matrice  $B$  in blocchi di righe. Diciamo  $R = \frac{M}{P} \text{ se } M \% P = 0, \frac{M}{P} + s \text{ altrimenti}$ . Ogni processo riceverà la matrice  $A_{loc} \in \mathbb{R}^{M \times R}$  e la matrice  $B_{loc} \in \mathbb{R}^{R \times L}$ . Ogni processo calcola una matrice  $C_{loc} \in \mathbb{R}^{M \times L}$  tramite il prodotto di  $A_{loc}$  e  $B_{loc}$ . Ogni processo esegue finalmente  $M \times L \times (R \text{ moltiplicazioni} + R - 3 \text{ somme})$  operazioni iniziali. In seguito per ottenere la matrice  $C$  finale, ogni processo dovrà sommare la propria matrice  $C_{loc}$  con tutte le altre al suo processo. Per fare ciò è possibile utilizzare 3 strategie della somma. Le più complete sono la seconda e la terza strategia. La quale permette di far arrivare il risultato finale ad un solo processo mentre la 3<sup>a</sup> permette di far arrivare il risultato finale a TUTTI i processi, salvaguardando comunque entrambe le strategie, le quali possono comunicare. Con la seconda strategia però, bisogna ricevere il risultato eseguire  $\log_2 P \times M \times L$  somme.

### Strategia 3

Le due matrici vengono distribuite a tutti i processi in blocchi di righe + blocchi di colonne = blocchi quadrati. Ogni processo deve avere possibilità di una figlia quadrata di dimensione  $\frac{M}{P} \times \frac{L}{P}$ . Ogni processo riceverà  $A_{loc} \in \mathbb{R}^{\frac{M}{P} \times \frac{M}{P}}$ ,  $B_{loc} \in \mathbb{R}^{\frac{L}{P} \times \frac{L}{P}}$ . Ogni processo può calcolare quindi  $C_{loc} \in \mathbb{R}^{\frac{M}{P} \times \frac{L}{P}}$ . Ogni processo esegue  $\frac{M}{P} \times \frac{L}{P} \times (\frac{M}{P} \times \frac{L}{P} \text{ moltiplicazioni} + \frac{M}{P} + 3 \text{ somme})$ . Ogni processo una volta calcolata la matrice necessita di ulteriori informazioni, ovvero deve poter calcolare la somma tra il suo  $C_{loc}$  ed i  $C_{loc}$  dei processi presenti sulla stessa riga, per fare ciò ogni processo sulla riga, invia a tutti i processi sulla riga il proprio  $A_{loc}$ , infine ogni processo sulla colonna della figlia invia al figlio processore sulla stessa colonna della figlia il proprio  $B_{loc}$ . Ogni processo potrà calcolare  $NP$  matrici di dimensione  $\frac{M}{P} \times \frac{L}{P}$ , compresa la sua. Ogni processo somma tutte queste matrici appena calcolate, salvaguardando almeno  $NP - 1$   $\times \frac{M}{P} \times \frac{L}{P}$  somme. Ogni processo invia al termine una componente della matrice finale, che se viene sommata fra tutti i processi, combinandole, si forma la matrice finale  $C$ .

3. Si enuncino le definizioni di:

- Efficienza

Per l'efficienza possiamo intendere Tante cose. L'efficienza è un algoritmo sequenziale dipende dal tipo di esecuzione delle  $T(N)$  operazioni f.p. Per il parallelismo non funziona così. In un algoritmo parallelo,

Per l'efficienza possiamo utilizzare Tante cose. L'efficienza è un algoritmo sequenziale di perfezione del tempo di esecuzione delle  $T(N)$  operazioni f.p. Per il parallelismo non funziona così. In un algoritmo parallelo il numero delle operazioni non dipende dal tempo di esecuzione, ma dal numero dei processori, poiché un calcolo parallelo impiega di eseguire più operazioni contemporaneamente. Per calcolare l'efficienza quindi a riferimento del speed up, dobbiamo riferirci alla relazione del tempo di esecuzione su processori rispetto al tempo di esecuzione su 1 processo.  $S(P) = \frac{T(1)}{T(P)}$  quindi poiché lo speed up mette in relazione le due tempi di calcolo, noi sappiamo che il tempo di esecuzione di  $T(1) = P \times T(P)$ , almeno dovrebbe essere così, pertanto se  $S(P) = \frac{P \times T(P)}{T(1)} \rightarrow S(P) = P$  vuol dire che si tratta di speedup ideale perché l'efficienza ideale che rappresenta il rapporto di  $S(P)$  su Pprocessori misura 1. L'efficienza misura quindi l'algoritmo sfrutta il parallelismo del calcolo.

- Si consideri il prodotto matrice per vettore  $A \cdot b = c$ , con  $A \in \mathbb{R}^{16 \times 8}$ ,  $b \in \mathbb{R}^8$  su un'architettura MIMD-DM, costituita da 4 processori, disposti secondo una griglia bidimensionale  $2 \times 2$ . La matrice A è distribuita per blocchi di righe e di colonne.
  - Che dimensione hanno i blocchi  $A_{loc}$  e come deve essere distribuito il vettore  $b$ ?
  - Si descriva l'algoritmo relativo alla risoluzione del problema.
  - Si calcolino speedup ed efficienza, secondo la definizione classica.

Vogliamo definire 3 strategie per calcolare il prodotto matrice vettore

### Strategia 1

La matrice A viene distribuita in blocchi di righe ed il vettore b viene sommato completamente ad ogni processo. Ogni processo riceverà: 1 primo 3x3 processori riceveranno  $A_{loc} \in \mathbb{R}^{6 \times 3}$  mentre l'ultimo processo riceverà  $A_{loc} \in \mathbb{R}^{5 \times 3}$ . I primi 3x3 processori calcoleranno  $6 \times (3 \times 3 \text{ moltip.} + 3 \times \text{ somme})$  mentre l'ultimo processo  $5 \times (3 \times 3 \text{ moltip.} + 3 \times \text{ somme})$ . Dopo averlo calcolato, il prodotto tra il proprio vettore b e la matrice  $A_{loc}$ . Il risultato sarà contenuto nel vettore  $c_{loc}$ . I primi 3x3 processori possiederanno  $c_{loc} \in \mathbb{R}^6$  mentre l'ultimo processo possiederà  $c_{loc} \in \mathbb{R}^5$ . Una volta terminato, il prodotto per ottenere il risultato finale. I primi processori dovranno inviare ad un solo processo il risultato, oppure inviare il risultato a tutti i processi se vogliano che ogni processo ottenga il risultato finale.

**Definizione Classica**  $T_{com} = 2T_{acc}$

$$T(36) = T_{acc} + T_{com}$$

Consideriamo la definizione classica, ovvero sappiamo che  $T(1) = T(P)$  sono uguali a  $T(1) = T_{acc} + T_{com}$ , sappiamo anche che  $T(1) = T_{acc}$  poiché non ci sono scambi di comunicazione.  $T(1) = T_{acc} = 35 \times (3 \times 3 \text{ moltip.} + 3 \times \text{ somme})$

$$T(1) = 2325 T_{acc} \quad T_{acc}(36) = 6 \times (3 \times 3 \text{ moltip.} + 3 \times \text{ somme}) = 650 T_{acc} \quad T_{com} = \frac{16 \times 6}{36} + \frac{5}{36} = 0,70 T_{com}$$

Invio di somme Invio di somma

$T(36) = 178 + 650 = 628 T_{acc}$  Sappiamo che lo speed up manda la relazione del tempo di esecuzione su p processi rispetto al tempo di esecuzione su 1 processo.  $S(36) = \frac{T(1)}{T(36)} = \frac{2325}{628} = 35,34$  L'efficienza misura il rapporto tra i tempi processi e il rapporto del rapporto tra  $\frac{S(36)}{P}$ .  $E(36) = \frac{35,34}{36} = 0,97$  Ricorriamo anche l'averaggio

Totale da misura quindi effettua lo speed up di tutto, cioè  $\Omega_{eff} = (P \times T(P)) - T(1) = 2323$ . Infine l'averaggio della comunicazione che rappresenta il rapporto tra il tempo di comunicazione ed il tempo calcolato per 1 processo.  $\Omega_{com} = \frac{T_{com}}{T_{acc}} = 0,39$  basta a fare **Grazie 22%**.

Per entrambi vogliamo eseguire tutte le operazioni in parallelo.

### Strategia 2

La matrice A viene data in blocchi di colonne, ed il vettore b viene distribuito per tutti i processi. I primi 10 processi riceveranno

$A_{loc} \in \mathbb{R}^{8 \times 2}$ ,  $b_{loc} \in \mathbb{R}^2$  e poi eseguiranno  $35(2+5)$  operazioni. Gli ultimi 6 processi riceveranno  $A_{loc} \in \mathbb{R}^{8 \times 2}$  e  $b_{loc} \in \mathbb{R}^3$ . Questi ultimi eseguiranno  $35(3+5)$  operazioni. Una volta calcolato il vettore  $c_{loc} \in \mathbb{R}^{26}$ , ogni processo dovrà sommare

$b_{loc} \in \mathbb{R}^3$ ,  $b_{loc} \in \mathbb{R}^3$ . I primi eseguiscono  $\rightarrow 1+2$  operazioni. I primi 6 processori ricevono  $Aloc \in \mathbb{R}^{24 \times 10}$  e  $Tcom = 7125$ . Quest'ultimo svolgono  $35(3FB)$  operazioni. Una volta calcolato il vettore  $loc \in \mathbb{R}^{36}$ , ogni processore somma i contributi  $\alpha_{loc}$  di ogni altro processore. Per fare ciò possono utilizzare 3 strategie di somma, le quali vengono consigliate rispettivamente alle alt. n. 2, al ca la seconda permette di far arrivare il risultato della somma ad un solo processore, e la Terza strategia permette di far arrivare il risultato a tutti i processori. Entrambe svolgono  $\log_2 P$  passi di comunicazione.

**Definizione Classica**  $Tcom = 2Tacc$

$$T(s) = Tacc(s) = 7125 \quad T(36) = Tacc + Tcom \quad Tacc = \underbrace{95 \times (3+2)}_{\text{Prodotti paralleli}} + \underbrace{\log_2 P \times 35}_{\text{Somme parallele}} = 675 + 380 = 855 + Tcom$$

$$Tcom = \log_2 P \times 35 = 380 Tcom \quad T(36) = 855 + 760 = \underbrace{1635}_{\text{Tacc}} + Tcom$$

$$\text{Ora } \frac{Tcom}{Tacc} = \frac{760}{855} = 0,88 \quad \text{Ora } f = 16 \times 0,88 - 7125 = 88735$$

$$S(36) = \frac{T(s)}{T(36)} = \frac{7125}{1635} = 4,46 \quad \bar{C}(36) = \frac{4,46}{36} = 0,127$$

**Vere e Vere generalizzata**

(a) **Vere**: si tratta di un tipo di ragionamento ovvero ovvero il tempo esecuzione di un algoritmo lo considerano come  $T_s$  Tempo sequenziale +  $T_p$  Tempo parallelo  $T(1) = T_s + T_p$  quindi se poniamo  $T_s = \alpha$   $T_p = 1 - \alpha$  abbiamo che

$$T(s) = 1 \quad \text{mentre} \quad S(P) = \frac{1}{\alpha + (1-\alpha)} \quad \text{a differenza dalla legge di Vere generalizzata, che considera}$$

oltre al parallelismo, costante la frazione di parallelismo ovvero  $\alpha$ : i quali rappresentano le operazioni eseguite in parallelo su i processi.

$$S(P) = \frac{1}{\alpha_1 + \sum_{i=2}^P \alpha_i} \quad \text{(Vere generalizzata): Ogni processo esegue insieme quindi } \alpha_{36} = 6 \times 95 \times (3+2) + 35 \times 35 \times (2+3)$$

$$\alpha_1 = 95 \text{ somme} \quad \alpha_2 = 2 \times 35 \quad \alpha_3 = 6 \times 35 \quad \alpha_4 = 8 \times 35 \quad \text{Totale} = 7125$$

$$\alpha_{36} = 2850 + 2850 = 5700$$

$$S(P) = \frac{1}{\frac{h \cdot 35}{7125} + \frac{5700}{7125}} = \frac{300}{35} = 8,67 \quad \bar{C}(36) = \frac{S(36)}{36} = 0,60$$

$$\text{Vere} \quad \alpha = \frac{95}{7125} \quad 1 - \alpha = \frac{9030}{7125} \quad S(36) = \frac{1}{\frac{95}{7125} + \frac{9030}{7125}} = \frac{60}{3} = 13,3 \quad \bar{C}(36) = \frac{S(36)}{36} = \frac{5}{6} = 0,83$$

$$T(P) = T_s + \frac{T_p}{P} + Tcom \rightarrow T(P) = 95 + \frac{1635}{36} + 760 = 955,93 \quad Tacc$$

**Strategia 3**

Ogni processore riceve una griglia  $\overline{P} \times \overline{P}$  quindi  $h \times h$ . Ogni processore riceve: i primi 6 processori ricevono vettore  $b_{loc} \in \mathbb{R}^{30}$  mentre gli ultimi 6 processori ricevono vettore  $b_{loc} \in \mathbb{R}^3$ . I processori presenti sulla griglia due righe ricevono quindi  $2 \times h$  processi ricevendo  $Aloc \in \mathbb{R}^{24 \times 10}$ . I processori sulla riga 3 ricevono  $Aloc \in \mathbb{R}^{24 \times 9}$  mentre i processori sulla ultima riga ricevono  $Aloc \in \mathbb{R}^{23 \times 8}$ . Ogni processore calcola il prodotto tra il proprio  $Aloc$  e  $b_{loc}$ , ma ogni processore somma il suo  $Aloc$  con tutti i  $b_{loc}$  dei processi presenti sulla stessa riga.

**Definizione Classica**  $Tcom = 2Tacc$

$$T(s) = 7125 Tacc \quad T(36) = Tacc + Tcom \quad Tacc = 24 \times (30 \text{ moltiplicazioni} + 3 \text{ somme}) + 3 \times 24 = 656 + 92 = 528 + Tcom$$

$$Tcom = 3 \times 24 = 72 Tcom \quad T(36) = 528 + 72 = 592 \quad \bar{C}(36) = \frac{592}{36} = 16,44$$

$$\bar{C}(36) = \frac{592}{36} = 0,66 \quad \text{Ora } \frac{16h}{328} = \frac{3}{35} = 0,27 \quad \text{Ora } f = 16 \times 0,27 - 7125 = 3624$$

**Vere generalizzata = Vere**

Ve lego ora eseguita  $\alpha_1 = 0$  operazioni e tutte le operazioni sono eseguite in parallelo ovvero.

$$\alpha_{36} = 2 \times 4 \times 2 \times (30 \text{ mult. pl. cariomi} + 3 \text{ somme}) + 4 \times 2 \times (3 \text{ mult. pl.} + 8 \text{ somme}) + 4 \times 2^3 \times (3m + 8s) = \\ = 3648 + 1632 + 1564 = 6844 + \text{SOHNE } 3 \times 4 \times 3 \times 2^4 + 4 \times 3 \times 2^3 = 864 + 286 = 1150$$

784

2. Si enunci la Legge di Ware Generalizzata, descrivendo nel dettaglio tutti i parametri.

Per enunciare la Legge di Ware Generalizzata, dobbiamo prima descrivere i componenti che la compongono. Partiamo da due che per misurare la complessità di un algoritmo separabile e' sufficiente calcolare (pero') i tempi di ogni operazione floating point. Il loro totale rappresenta la complessità di tempo di un algoritmo separabile. Per quanto riguarda l'algoritmo parallelo tutto ciò non è possibile, poche ad un punto di tempo vengono eseguite contemporaneamente più operazioni alla volta. Dunque Ware ha stimato che i tempi di esecuzione di un algoritmo parallelo su  $p$  processori dipendono da due componenti:

$T_s$  tempo per eseguire le parti separabili +  $T_p$  tempo di esecuzione parallela. Se consideriamo  $T(s) = 1$   $T_s = \alpha$  e  $\alpha - \alpha = T_p$ , seppiamo che lo speed up misura la riduzione del tempo di esecuzione su  $P$  processori, rispetto al tempo di esecuzione su 1 solo processore.

$$S(P) = \frac{T(1)}{T(P)} \rightarrow \text{Se poniamo } T(P) = T_s + T_p = S(P) = \frac{1}{\alpha + \frac{1-\alpha}{P}} \text{ abbiamo poi l'andamento totale che misura quanto dista lo speed up da quello ideale. } \Omega_h(T(P)) = (P \times T(P))^{-T(s)} \rightarrow \left( P \times \left( \alpha + \frac{1-\alpha}{P} \right) - 1 \right). \text{ L'efficienza misura misura il parallelismo di un algoritmo, quindi misura il rapporto tra speed up e numero di processori. } E(P) = \frac{S(P)}{P} \rightarrow E(P) = \frac{1}{\alpha + \frac{1-\alpha}{P} \times P} = \frac{1}{\alpha + \frac{1}{P}} \text{ Per quanto riguarda la legge di Ware Generalizzata, poiché Ware assume che ogni algoritmo possa lavorare in sequenziale e/o in parallelo, non assume il numero di processori che lavorano contemporaneamente. Per Ware generalizzata } \alpha_3 = T_s \text{ mentre esistono tanti } \alpha_i \text{ quanti sono i processori che lavorano contemporaneamente sul ogni punto. } \bar{T}(P) = \alpha_3 + \sum_{i=2}^P \frac{\alpha_i}{i} \text{ quindi } S(P) = \frac{1}{\alpha_3 + \sum_{i=2}^P \frac{\alpha_i}{i}}.$$

4. Si consideri la seguente routine della libreria MPI:

`MPI_Cart_create(MPI_Comm comm_old, int dim, int *ndim, int *period,  
int *reorder, MPI_Comm *new_comm);`

- Cosa è possibile effettuare utilizzando tale routine?
- Descrivere, in dettaglio, i parametri.

Con `(P)-CART-CREATE` è possibile creare un nuovo communicator, partendo dal vecchio. È possibile continuare una griglia virtuale esistente di processori, definendo la dimensione, la periodicità e anche la multidimensionalità. Essa è un'operazione collettiva.

- `comm-old` vecchio communicator
- `dim` numero di dimensioni della griglia
- `*ndim` vettore contenente le lunghezze di ciascuna dimensione.
- `*period` vettore di dimensioni che contiene la periodicità di ogni dimensione.
- `reorder` permette di riordinare i numeri dei processori.
- `new-comm` comunicatore di output.

4. Mostrare come si apre una regione parallela in OPEN-MP e illustrare lo schema di fork join.

OpenMP è un API che si basa su architettura SIMD-SR. Sfrutta la memoria condivisa per eseguire gli algoritmi paralleli. Come sappiamo alle istruzioni di un algoritmo possono essere più di tre che vengono eseguite in parallelo e non si sovrappone le regioni eseguite in sequenziale. OpenMP sfrutta un parallelismo multi-threading ovvero, l'algoritmo parte in maniera sequenziale eseguito principalmente da un solo Thread detto Thread master, per aprire una regione parallela viene eseguito il comando #pragma omp define anche "critico per compilazione" da qui nasce la regione parallela ed avviene una fork, ovvero il Thread master divide l'esecuzione su più Thread i quali lavoreranno parallellamente. Una volta terminato il crittico viene effettuata una join, ovvero tutti i dati vengono sincronizzati, (quelli contenuti) e ritorna l'esecuzione avanti Thread master. In aggiunta al crittico vengono definite le direttive per il completamento: master, critical, parallel, barrier. Ci sono anche le directive Worksharing, che permettono di lavorare in modo ottimizzato o per un uso più frequentale: for, single, sections. Infine vengono eseguite chiuse .

5. Si descrivano le problematiche dell'algoritmo della somma di N numeri su un'architettura MIMD a memoria condivisa, spiegando come si risolvono utilizzando OpenMP.



Semafori

I Semafori sono primitive implementate attraverso contatori, fornite dal sistema operativo, per controllare l'accesso alle risorse condivise fra più processi (threads) che condividono la medesima memoria

Sono utilizzati come meccanismi di bloccaggio per evitare che altri threads accedano alla memoria condivisa contemporaneamente al thread che la sta già utilizzando.