...Scientific Software will lead mathematics
to focus more heavily on
problem solving and algorithms ...
J. Rice, 1998

...Il Software Scientifico condurrà la Matematica
a porre sempre di piú l'attenzione
sugli algoritmi per la risoluzione di problemi ...

Capitolo 4

I parametri di valutazione del software parallelo

La varietà e la complessità delle architetture esistenti ha accresciuto notevolmente l'influenza dell'ambiente di calcolo sullo sviluppo, e quindi sulla valutazione di algoritmi e software[1]. La struttura SIMD o MIMD, la presenza di una o piú memorie condivise o distribuite, lo schema di connessione e la velocità di comunicazione tra i processori, il numero di processori, l'architettura e la potenza di ciascuno di essi, sono solo alcuni dei fattori che influenzano le prestazioni di un algoritmo in ambiente di calcolo parallelo.

Le funzioni classiche relative alla complessità di tempo e alla complessità di spazio utilizzate in ambito sequenziale, non sono piú adeguate alla valutazione degli algoritmi paralleli, è necessario individuare un insieme di parametri, che richiede revisioni e modifiche a mano a mano che i sistemi di calcolo evolvono.

♣ Esempio 12

Consideriamo la somma di 16 numeri. Indichiamo con p il numero di processori, con

[1] Nel seguito di questo capitolo spesso useremo il termine *algoritmo* riferendoci alla sua implementazione nell'ambiente di calcolo.

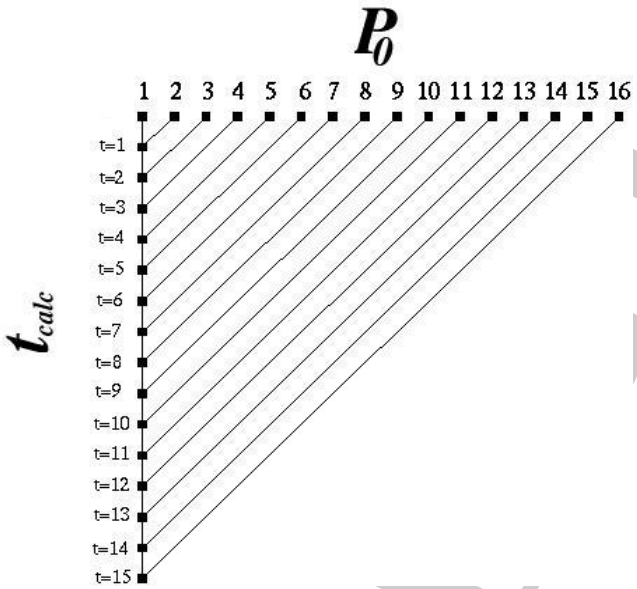

Figura 4.1: *I dati sono memorizzati su un processore. Il tempo di esecuzione è* 15 t_{calc}

T_p il tempo di esecuzione dell'algoritmo su p processori e con t_{calc} il tempo necessario all'esecuzione di una operazione di addizione.

Se $p = 1$ il numero di addizioni eseguite è 15, quindi il tempo di esecuzione è $T_1 = 15\ t_{calc}$ (Fig. 4.1).

Se i processori sono due, $p = 2$, il numero di addizioni eseguite è sempre 15, ma $T_2 = 8\ t_{calc}$ (Fig. 4.2). I dati sono infatti distribuiti tra i due processori. A ciascun processore vengono assegnati 8 numeri e ciascun processore calcola 7 somme, impiegando un tempo di 7 t_{calc}. Le somme parziali sono poi sommate in 1 t_{calc}; il tempo complessivo di calcolo è 8 t_{calc}.

Se $p = 4$, ogni processore possiede 4 numeri ed esegue 3 addizioni in un tempo 3 t_{calc} per ottenere il risultato parziale; le quattro somme parziali ottenute saranno eseguite in un tempo 2 t_{calc} cosí da ottenere il risultato finale in un tempo $T_4 = 5\ t_{calc}$ (Fig. 4.3).

Se invece $p = 8$, ogni processore possiede 2 numeri ed esegue 1 somma in 1 t_{calc}. Quindi quattro coppie di processori sommano i risultati parziali in 1 t_{calc}. Al passo successivo due coppie di processori sommano i risultati parziali in 1 t_{calc} e infine una coppia di processori ottiene il risultato finale in 1 t_{calc} (Fig. 4.4). In totale, $T_8 = 4\ t_{calc}$.

In definitiva, per la somma di sedici numeri sono necessari i tempi di calcolo indicati nella Tab. 4.1, in cui è stato assunto t_{calc} come unità di tempo.

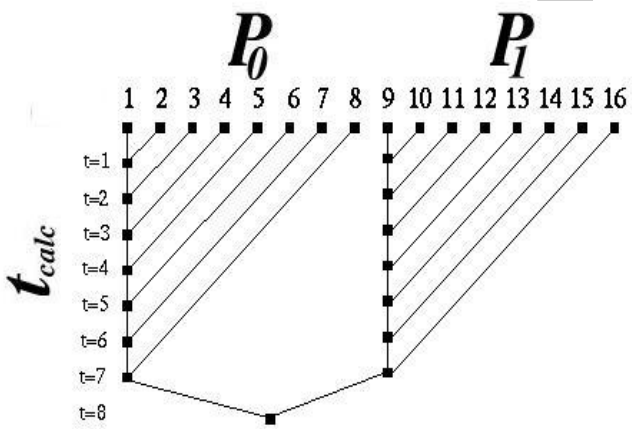

Figura 4.2: *Schema esecutivo dell'algoritmo per il calcolo della somma di* 16 *numeri con* $p = 2$ *processori: i dati vengono distribuiti su 2 processori. Il tempo di esecuzione è* 8 t_{calc}

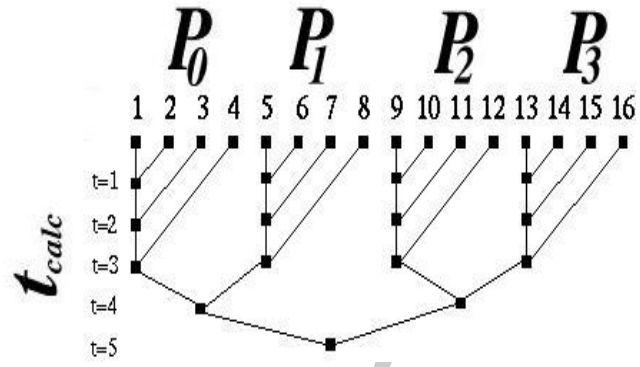

Figura 4.3: *Schema esecutivo dell'algoritmo per il calcolo della somma di* 16 *numeri con* $p = 4$ *processori: i dati vengono distribuiti su 4 processori. Il tempo di esecuzione è* 5 t_{calc}

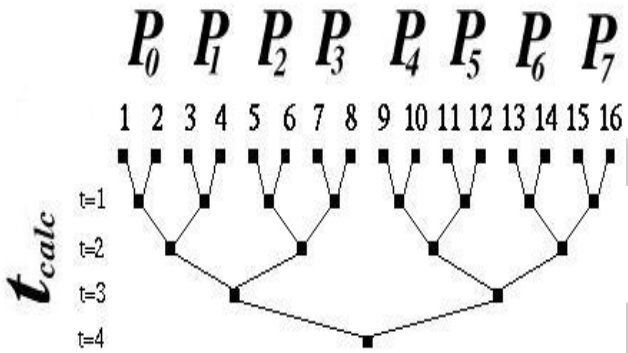

Figura 4.4: *Schema esecutivo dell'algoritmo per il calcolo della somma di* 16 *numeri con* $p = 8$ *processori: i dati vengono distribuiti su* 8 *processori. Il tempo di esecuzione è* 4 t_{calc}

p	T_p
1	15 t_{calc}
2	8 t_{calc}
4	5 t_{calc}
8	4 t_{calc}

Tabella 4.1: *Nelle colonne sono indicati p, il numero di processori, e* T_p*, il tempo di esecuzione.*

Osserviamo che, in generale, se i processori sono $p = 2^s$, il tempo di esecuzione T_p si puó esprimere come

$$T_p = (\frac{n}{p} - 1 + log_2 p)\ t_{calc}$$

Ci chiediamo allora quale sia l'algoritmo parallelo che impiega meno tempo e quanto questo sia piú veloce di quello sequenziale. A tal fine si introduce il concetto di *speed up* [2].

△

Indicato con T_p il tempo di esecuzione dell'algoritmo su p processori, ci aspettiamo che se $p = 2$, T_1 sia il doppio di T_2, cioè $T_1/T_2 = 2$. Ancora, se $p = 4$, ci aspettiamo che T_1 sia il quadruplo

[2] Qui e nel seguito assumiamo che tuttti i processori siano omogenei.

di T_4, cioè $T_1/T_4 = 4$. Se si hanno a disposizione p processori, l'obiettivo è impiegare $1/p$ volte il tempo necessario a risolverlo con un solo processore.

Definizione: *se T_1 è il tempo di esecuzione di un algoritmo su 1 processore e T_p è il tempo di esecuzione su p processori, lo speed up S_p è definito cosí*:

$$S_p = \frac{T_1}{T_p} \tag{4.1}$$

Lo speed up misura la riduzione del tempo di esecuzione rispetto all'algoritmo su un processore.

Dalla definizione di speed up segue che $S_p \leq p$ ([3]). Si definisce quindi *lo speed up ideale*:

$$S_p^{ideale} = p$$

cioé l'algoritmo parallelo risulta migliore quanto piú S_p è prossimo a p. Nel caso *ideale* si ha che

$$S_p = p \Leftrightarrow T_1 = p \cdot T_p$$

mentre in generale:

$$S_p < p \Rightarrow T_1 < pT_p$$

da cui deriva in modo naturale la seguente

Definizione:*Si definisce Overhead totale*

$$O_h = pT_p - T_1 \tag{4.2}$$

[3]Sebbene esistano casi in cui può accadere che $S_p > p$. In tal caso si parla di speed up *"superlineare"*. Lo speed up superlineare puó dipendere ad esempio da accessi alla memoria non efficienti, a causa delle quali $T_1 > pT_p$ ..

con

$$O_h > 0$$

L'Overhead misura quanto lo speed up effettivo differisca da quello ideale.

♣ Esempio 13

Ritornando all'esempio della somma di sedici numeri si ha la tabella seguente:

p	$\mathbf{T_p}$	$\mathbf{S_p}$	$\mathbf{S_p^{ideale}}$	$\mathbf{O_h}$
1	$15\ t_{calc}$	1.00	1	0
2	$8\ t_{calc}$	1.88	2	1
4	$5\ t_{calc}$	3.00	4	5
8	$4\ t_{calc}$	3.75	8	14

Tabella 4.2: *Nelle colonne sono indicati p, il numero di processori, T_p, il tempo di esecuzione, S_p , lo speed up, S_p^{ideale}, lo speed up ideale e O_h, l'overhead.*

Poiché per l'algoritmo della somma di n numeri $T_1 = n - 1$ e $T_p = \frac{n}{p} - 1 + log_2 p$, l'espressione dell'overhead in questo caso é:

$$O_h = 1 - p + plog_2 p \tag{4.3}$$

Dalla Tab. 4.2 è evidente che con 8 processori si è avuta la riduzione maggiore del tempo di esecuzione. Quindi se si vuole calcolare la somma di 16 numeri nel tempo minore l'algoritmo su 8 processori é da preferire. Da un'analisi piú attenta ci accorgiamo però che sebbene in questo caso quanti piú processori si usano tanto piú si riduce il tempo di esecuzione, ciò non sempre significa che l'ambiente di calcolo sia stato utilizzato in maniera efficace e questo é confermato dall'aumento dell'overhead totale. Lo speed up su 2 processori infatti è *"il piú vicino"* allo speed up ideale corrispondente, quindi sembrerebbe, in questo caso, che il miglior uso dei processori si abbia per $p = 2$.

Da questo esempio appare chiaro che accanto alla misura dello speed up conviene avere una informazione di quanto siano state

utilizzate le risorse di calcolo. In altre parole, bisogna *"rapportare"* lo speed-up al numero di processori.

Definizione: *Sia p il numero di processori e S_p lo speed-up ad esso relativi. Si definisce efficienza E_p il parametro*

$$E_p = \frac{S_p}{p} \tag{4.4}$$

che fornisce un'indicazione di quanto sia stato utilizzato il parallelismo del calcolatore..

Dalla definizione di speed up ed efficienza segue che

$$E_p \leq 1$$

Si definisce quindi l'*efficienza ideale*:

$$E_p^{ideale} = 1$$

cioé l'algoritmo parallelo risulta migliore quanto piú E_p è vicino ad 1.
Spesso é utile esprimere l'efficienza in funzione dell'Overhead totale:

$$E_p = \frac{1}{\frac{O_h}{T_1} + 1} \tag{4.5}$$

♣ Esempio 14

Nel caso della somma di 16 numeri su p processori si hanno i risultati riportati in Tabella 4.3.

Si osserva che per $p = 2$ il parametro efficienza è piú prossimo ad 1. Quindi, l'algoritmo per il calcolo della somma di 16 numeri su $p = 2$ processori è quello che sfrutta meglio il parallelismo.

p	E_p
1	1.00
2	0.94
4	0.75
8	0.47

Tabella 4.3: *Nelle colonne sono indicati p, il numero di processori utilizzati, e E_p, l'efficienza.*

In generale in ogni algoritmo il tempo di esecuzione sequenziale T_1 è costituito da una *parte seriale* (relativa alle operazioni che sono eseguite esclusivamente in sequenziale) e da una *parte parallela* (relativa all'insieme di operazioni che possono essere eseguite concorrentemente):

$$T_1 = \alpha T_1 + (1-\alpha)T_1$$

Allora il tempo di esecuzione dell'algoritmo parallelo T_p sará costituito dalla stessa parte seriale αT_1 e dalla restante parte $(1-\alpha)T_1$, ovviamente divisa per il numero dei processori, cioé:

$$T_p = \alpha \cdot T_1 + \frac{(1-\alpha)T_1}{p}$$

e quindi:

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\alpha \cdot T_1 + \frac{(1-\alpha)T_1}{p}} = \frac{1}{\alpha + \frac{(1-\alpha)}{p}}$$

per cui sussiste la seguente:

Definizione: *Se α è la frazione sequenziale del tempo T_1 e $(1-\alpha)$ è la frazione parallela del tempo T_1, si ha il modello teorico di speed*

up seguente

$$S_p = \frac{1}{\alpha + \frac{(1-\alpha)}{p}}$$

noto come legge di Ware [53] o di Amdhal.

♣ Esempio 15

Si consideri la somma di N numeri su p processori. In particolare sia $N = 4$ e $p = 2$.

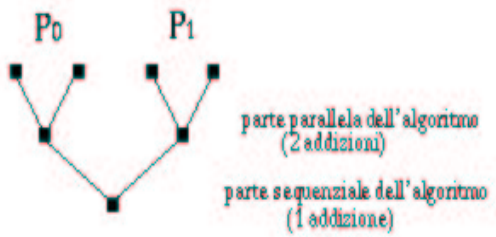

Figura 4.5: *Nel grafo dell'algoritmo della somma di N numeri è evidenziata la parte parallela e quella sequenziale.*

Poiché le addizioni sono 3, ($T_1 = 3\ t_{calc}$) indichiamo con

$$\alpha = \frac{1}{3}$$

la parte di T_1 relativa alle operazioni eseguite sequenzialmente e con

$$\beta = 1 - \alpha = \frac{2}{3}$$

la parte di T_1 relativa alle operazioni eseguite in parallelo dai 2 processori.
Pertanto, il tempo T_2 si puó esprimere in termini di T_1 come:

$$T_2 = \frac{1}{3}T_1 + \frac{\frac{2}{3}T_1}{2}$$

e

$$S_2 = \frac{T_1}{T_2} = \frac{T_1}{\frac{1}{3}T_1 + \frac{\frac{2}{3}T_1}{2}} = \frac{3}{2}$$

△

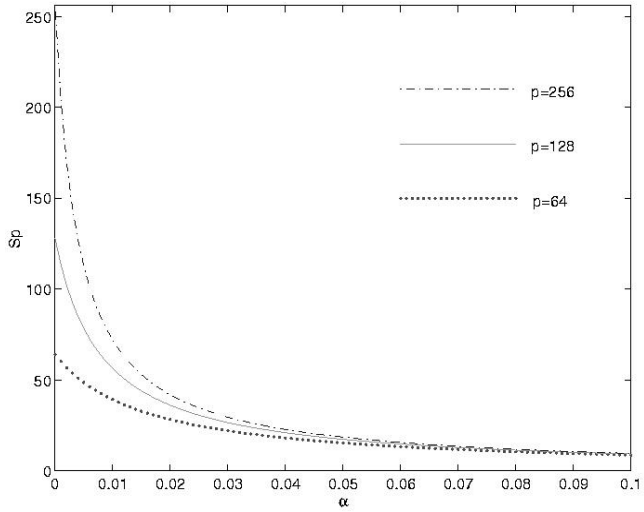

Figura 4.6: *In figura è riportato il valore dello speed up in funzione di α per i valori di $p = 64, 128, 256$*

Assumendo che α sia costante rispetto a p, poiché:

$$\lim_{p\to\infty} \frac{1-\alpha}{p} = 0$$

risulta

$$S_p \leq \frac{1}{\alpha}$$

Ovvero dalla legge di Amdhal segue che la parte sequenziale può degradare fortemente lo speed up quando il numero dei processori é sufficientemente elevato (vedi Fig. 4.6).

♣ Esempio 16

Consideriamo la somma di $N = 8$ numeri su $p = 4$ processori (Fig. 4.7).

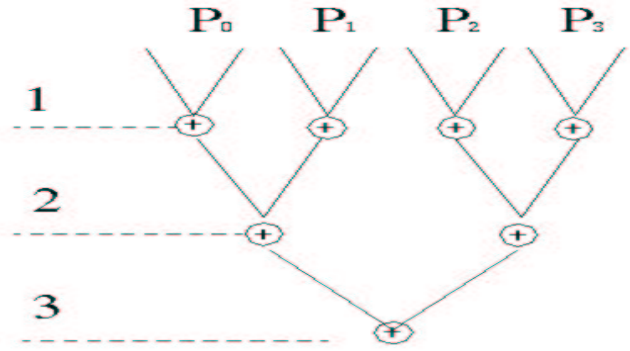

Figura 4.7: Schema dell'algoritmo su 4 processori che eseguono la somma di N numeri

1. Al passo 1 vengono eseguite 4 addizioni da 4 processori.
2. Al passo 2 vengono eseguite 2 addizioni da 2 processori.
3. Al passo 3 viene eseguita 1 addizione da 1 processore.

Al passo 1 é evidente la parte parallela dell'algoritmo, in cui vengono eseguite le operazioni concorrentemente da tutti i processori. Al passo 3 é presente la parte sequenziale, in cui le operazioni vengono eseguite da un solo processore. Cerchiamo di definire il contributo del passo 2 in cui vengono eseguite operazioni solo da un certo numero di processori $\tilde{p} < p = 4$.

Nel nostro caso risulta

$$T_1 = 7\ t_{calc}$$

Al passo 1 la frazione di T_1 relativa alle 4 addizioni eseguite dai 4 processori è

$$\alpha_4 = \frac{4}{7}$$

e il contributo a T_4, tempo di esecuzione su 4 processori, è

$$\alpha_4 \cdot \frac{T_1}{4}$$

Al passo 3 la frazione di T_1 relativa all'addizione eseguita da 1 processore è

$$\alpha_1 = \frac{1}{7}$$

e il contributo a T_4 è

$$\alpha_1 \cdot T_1$$

Consideriamo quindi le 2 addizioni eseguite dai 2 processori al passo 2. Risulta

$$\alpha_2 = \frac{2}{7}$$

e il contributo a T_4 è

$$\alpha_2 \cdot \frac{T_1}{2}$$

In generale, indicato con i il numero di processori che contribuiscono ad α_i si ha

$$T_p = \sum_{i=1}^{p} \alpha_i \frac{T_1}{i}$$

dove T_p rappresenta il tempo di esecuzione relativo alle operazioni eseguite da tutti i processori.

Nel caso in cui $p = 4$ e $T_1 = 7$ si ha:

$$T_4 = \alpha_1 T_1 + \alpha_2 \frac{T_1}{2} + \alpha_4 \frac{T_1}{4} = \frac{1}{7} \cdot 7 + \frac{2}{7} \cdot \frac{7}{2} + \frac{4}{7} \cdot \frac{7}{4} = 3$$

△

Detta α_i la frazione di T_1 eseguita da i processori ($1 \leq i \leq p$) la formulazione della legge di Ware piú generale è la seguente:

$$S_p = \frac{T_1}{\alpha_1 T_1 + \sum_{k=2}^{p-1} \alpha_k \cdot \frac{T_1}{k} + \alpha_p \cdot \frac{T_1}{p}} = \frac{1}{\alpha_1 + \sum_{k=2}^{p-1} \frac{\alpha_k}{k} + \frac{\alpha_p}{p}}$$

♣ Esempio 17

Consideriamo la somma di $N = 32$ numeri su $p = 2$ processori come mostrato in Fig. 4.8/A.

Risulta:

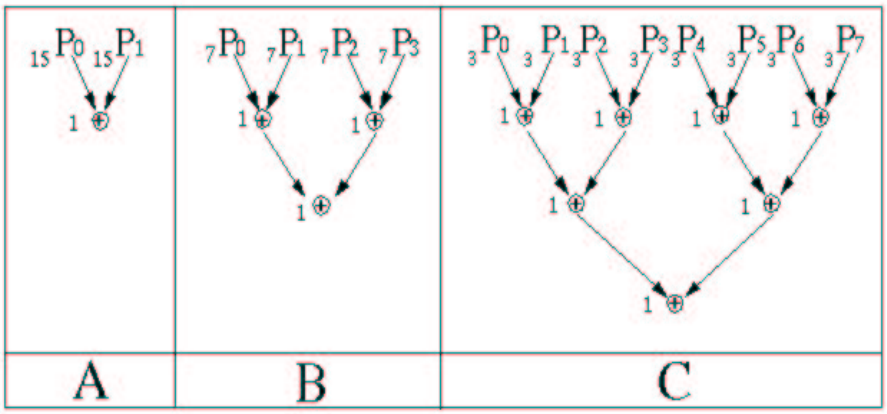

Figura 4.8: *Schematizzazione della somma di $N = 32$ numeri rispettivamente su $p = 2$ processori, $p = 4$ processori e $p = 8$ processori. Ad ogni passo è indicato il numero di somme effettuate da ciascun processore.*

$$T_1 = N - 1 = 31$$

$$\alpha_1 = \frac{1}{31} = 0,03 \qquad \alpha_2 = \frac{30}{31} = 0.96$$

$$S_2 = \frac{1}{\alpha_1 + \frac{\alpha_2}{2}} = \frac{1}{\frac{1}{31} + \frac{30}{31} \cdot \frac{1}{2}} = \frac{31}{16} = 1,9$$

$$E_2 = \frac{S_2}{p} = \frac{1,9}{2} = 0,95$$

Consideriamo ora la somma di $N = 32$ numeri su $p = 4$ processori come mostrato in Fig. 4.8/B.
Risulta:

$$T_1 = N - 1 = 31$$

$$\alpha_1 = \frac{1}{31} \qquad \alpha_2 = \frac{2}{31} \qquad \alpha_4 = \frac{28}{31}$$

$$S_4 = \frac{1}{\alpha_1 + \frac{\alpha_2}{2} + \frac{\alpha_4}{4}} = \frac{1}{\frac{1}{31} + \frac{2}{31} \cdot \frac{1}{2} + \frac{28}{31} \cdot \frac{1}{4}} = \frac{1}{\frac{1}{31} + \frac{1}{31} + \frac{7}{31}} = \frac{31}{9} = 3,4$$

$$E_4 = \frac{S_4}{p} = \frac{3,4}{4} = 0,85$$

Consideriamo ora la somma di $N = 32$ numeri su $p = 8$ processori come mostrato in Fig. 4.8/C.
Risulta:

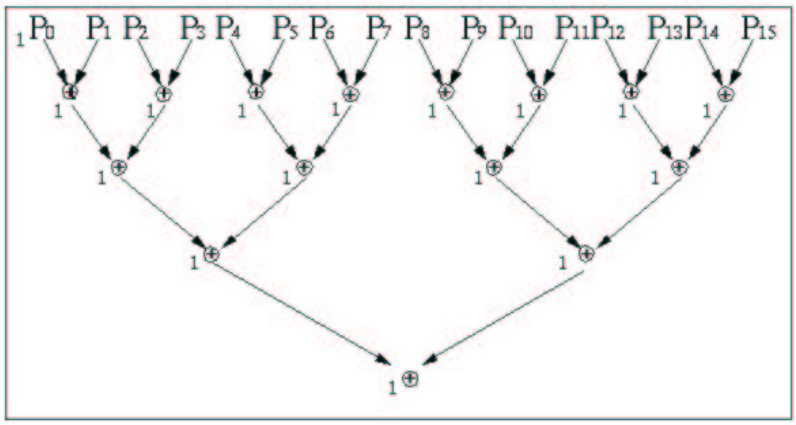

Figura 4.9: *Schematizzazione della somma di $N = 32$ numeri su $p = 16$ processori. Ad ogni passo è indicato il numero di somme effettuate da ciascun processore.*

$$T_1 = N - 1 = 31$$

$$\alpha_1 = \frac{1}{31} \qquad \alpha_2 = \frac{2}{31} \qquad \alpha_4 = \frac{4}{31} \qquad \alpha_8 = \frac{(31-7)}{31} = \frac{24}{31}$$

$$S_8 = \frac{1}{\alpha_1 + \sum_{k=2}^{7} \frac{\alpha_k}{k} + \frac{\alpha_8}{8}} = \frac{1}{\frac{1}{31} + \frac{2}{31}\cdot\frac{1}{2} + \frac{4}{31}\cdot\frac{1}{4} + \frac{24}{31}\cdot\frac{1}{8}} = \frac{1}{\frac{1}{31} + \frac{1}{31} + \frac{1}{31} + \frac{3}{31}} = \frac{31}{6} = 5,1$$

$$E_8 = \frac{S_8}{p} = \frac{5,1}{8} = 0,6$$

Consideriamo ora la somma di $N = 32$ numeri su $p = 16$ processori, come mostrato in Fig. 4.9.
Risulta:

$$T_1 = N - 1 = 31$$

$$\alpha_1 = \frac{1}{31} \qquad \alpha_2 = \frac{2}{31} \qquad \alpha_4 = \frac{4}{31} \qquad \alpha_8 = \frac{8}{31} \qquad \alpha_{16} = \frac{(31-15)}{31} = \frac{16}{31}$$

$$S_{16} = \frac{1}{\alpha_1 + \sum_{k=2}^{15} \frac{\alpha_k}{k} + \frac{\alpha_{16}}{16}} = \frac{1}{\frac{1}{31} + \frac{2}{31}\cdot\frac{1}{2} + \frac{4}{31}\cdot\frac{1}{4} + \frac{8}{31}\cdot\frac{1}{8} + \frac{16}{31}\cdot\frac{1}{16}} = \frac{1}{\frac{1}{31} + \frac{1}{31} + \frac{1}{31} + \frac{1}{31} + \frac{1}{31}} = \frac{31}{5} = 6,2$$

$$E_{16} = \frac{S_{16}}{p} = \frac{6,2}{16} = 0,3$$

Quindi, se $N = 32$ si ha:

Si osserva che il peso della frazione $1 - \alpha_p$ aumenta all'aumentare del numero di processori. Se N è fissato, al crescere del numero di processori, la quantità $1 - \alpha_p$ aumenta facendo cosí degradare speed-up ed efficienza.

△

Come si può osservare dagli esempi precedenti, se la dimensione

$\mathbf{p}$	α_1	$\alpha_2 + \ldots + \alpha_{p-1}$	α_p	$\mathbf{S_p}$	$\mathbf{E_p}$
2	0,032	0	0,968	1,9	0,95
4	0,032	0,065	0,903	3,4	0,85
8	0,032	0,193	0,775	5,1	0,6
16	0,032	0,452	0,516	6,2	0,3

Tabella 4.4: *Nelle colonne sono indicati p, il numero di processori, α_1, la frazione sequenziale di T_1, $\sum_{k=2}^{p-1} \alpha_k$, la frazione parallela di T_1 eseguita da k processori, α_p, la frazione parallela di T_1 eseguita da p processori, oltre che S_p, lo speed up, ed E_p, l'efficienza.*

del problema è fissata, al crescere del numero di processori non solo non si riescono ad ottenere speed-up vicini a quello ideale, ma le prestazioni peggiorano.

Vediamo allora cosa accade alla parte sequenziale e alla parte parallela, e allo speed-up e all'efficienza, se manteniamo fisso il numero di processori mentre la dimensione N del problema aumenta.

♣ Esempio 18

Consideriamo la somma di N numeri su $p = 2$ processori.

Per $N = 8$ risulta $\alpha_1 = \frac{1}{7}$, $S_2 = \frac{7}{4} = 1,75$, $E_2 = \frac{1,75}{2} = 0,875$

Per $N = 16$ risulta $\alpha_1 = \frac{1}{15}$, $S_2 = \frac{15}{8} = 1,8$, $E_2 = \frac{1,8}{2} = 0,9$

Per $N = 32$ risulta $\alpha_1 = \frac{1}{31}$, $S_2 = \frac{31}{16} = 1,9$, $E_2 = \frac{1,9}{2} = 0,96$

Per $N = 64$ risulta $\alpha_1 = \frac{1}{63}$, $S_2 = \frac{63}{32} = 1,96$, $E_2 = \frac{1,96}{2} = 0,99$

$\mathbf{N}$	α_1	α_2	$\mathbf{S_2(N)}$	$\mathbf{E_2(N)}$
8	0,14	0,86	1,75	0,875
16	0,06	0,93	1,8	0,9
32	0,03	0,96	1,9	0,96
64	0,01	0,98	1,96	0,99

Tabella 4.5: *Nelle colonne sono indicati N, la dimensione del problema, α_1, la frazione sequenziale di T_1, e α_2, la frazione parallela di T_1, $S_2(N)$, lo speed up, e $E_2(N)$, l'efficienza.*

Come si osserva dalla Tab. 4.5, aumentando la dimensione del problema, la parte sequenziale α_1 va a zero.

$\triangle$

Dall'esempio precedente si evidenzia come al crescere della dimensione del problema la frazione sequenziale dell'algoritmo parallelo decresca significativamente all'aumentare del numero di processori. Dunque, se p è fissato ed $N \to \infty$, la parte sequenziale α tende a zero[4]. (vedi Tab. 4.6):

$$\alpha + \frac{1-\alpha}{p} \to \frac{1}{p} \quad e \quad S_p \to p$$

In effetti, la parte sequenziale α di un algoritmo, è funzione sia della dimensione del problema sia del numero di processori, cioè

$$\alpha = \alpha(N, p)$$

con

$$\alpha = \alpha(N,p) \to 0 \; se \begin{cases} N \to \infty \\ p = p_0 \end{cases}$$

$$\alpha = \alpha(N,p) \to \infty \; se \begin{cases} p \to \infty \\ N = N_0 \end{cases}$$

Pertanto, é possibile ottenere speed-up prossimi a quello ideale solo se si fissa p e si fa crescere N; d'altro canto però non è possibile aumentare N in maniera arbitraria perché le risorse (hardware) disponibili sono limitate. In altre parole, la legge di Amdhal, sembra

[4] A tal proposito si ricorda la definizione data da C. Moler, che considera un'applicazione "effettivamente parallela" quando la sua frazione sequenziale α è piccola.

fornire un modello attendibile solo se $p \leq p_0$ e $N \leq N_0$, con N_0 e p_0 opportunamente prefissati.

Cosa succede, allora, se, aumentando il numero di processori p, utilizziamo lo stesso algoritmo per risolvere lo stesso problema di dimensione maggiore?

♣ Esempio 19

Vediamo cosa accade se aumenta sia la dimensione N del problema sia il numero p di processori. In particolare aumentiamo si N che p in modo che N/p sia costante.

Per $N = 8$ e $p = 2$ risulta
$\alpha_1 = \frac{1}{7} = 0,14$, $\alpha_2 = \frac{6}{7} = 0,85$,
$S_2(8) = \frac{7}{4} = 1,75$, $E_2(8) = \frac{1,75}{2} = 0,875$

Per $N = 16$ e $p = 4$ risulta
$\alpha_1 = \frac{1}{15} = 0,06$, $\alpha_4 = \frac{12}{15} = 0,8$,
$S_4(16) = \frac{15}{5} = 3$, $E_4(16) = \frac{3}{4} = 0,75$

Per $N = 32$ e $p = 8$ risulta
$\alpha_1 = \frac{1}{31} = 0,03$, $\alpha_8 = \frac{24}{31} = 0,77$,
$S_8(32) = \frac{31}{6} = 5,1$, $E_8(32) = \frac{5,1}{8} = 0,64$

Per $N = 64$ e $p = 16$ risulta
$\alpha_1 = \frac{1}{63} = 0,01$, $\alpha_{16} = \frac{48}{63} = 0,76$,
$S_{16}(64) = \frac{63}{7} = 9$, $E_{16}(64) = \frac{9}{16} = 0,56$

$\mathbf{N/p = 4}$	$\alpha_\mathbf{1}$	$\alpha_\mathbf{p}$	$\mathbf{S_p(N)}$	$\mathbf{E_p(N)}$
$N = 8\, p = 2$	$0,14$	$0,86$	$1,75$	$0,875$
$N = 16\, p = 4$	$0,06$	$0,8$	3	$0,75$
$N = 32\, p = 8$	$0,03$	$0,77$	$5,1$	$0,64$
$N = 64\, p = 16$	$0,01$	$0,76$	9	$0,56$

Tabella 4.6: *Nella prima colonna sono indicati N, la dimensione del problema, e p, il numero di processori; nelle colonne successive α_1, la frazione sequenziale di T_1, α_p, la frazione parallela di T_1, $S_p(N)$, lo speed up ed $E_p(N)$, l'efficienza.*

Dalla Tab. 4.6 risulta che, aumentando sia la dimensione del problema sia il numero di processori, rimane *"quasi costante"* la parte parallela mentre decresce la parte sequenziale. Inoltre, le prestazioni dell'algoritmo non degradano.

△

♣ Esempio 20

Si supponga di risolvere un problema di dimensione $N = 2$ su $p = 1$ processore in un tempo t. Ci si aspetta di risolvere nello stesso tempo t:

- su $p = 2$ processori un problema di dimensione $N = 4 = 2 * 2$;
- su $p = 4$ processori un problema di dimensione $N = 8 = 2 * 4$;
- su $p = 8$ processori un problema di dimensione $N = 16 = 2 * 8$,

ovvero che

$$T_1(N) = T_2(2N) = T_4(4N) = \ldots = T_p(Np)$$

cioè che all'aumentare del numero di processori, nello stesso tempo riesco a risolvere il problema di dimensioni piú grandi. Quindi:

$$1 = \frac{T_1(N)}{T_p(Np)} \leftrightarrow p = p\frac{T_1(n)}{T_p(Np)}$$

assumiamo allora che

$$p \times T_1(N) = T_p(Np)$$

da cui

$$\frac{T_1(Np)}{T_p(Np)} = p$$

ovvero, otteniamo in questo caso lo speed up ideale.

△

Nell'esempio 20 abbiamo assunto che

$$T_1(N\ p) \simeq p \times T_1(N)$$

cioè si assume $T_1(N)$ circa uguale al tempo che si ottiene moltiplicando per p il tempo per risolvere su 1 processore il problema di

dimensione N. In generale, se

$$N = p \times N_{loc} \ ,$$

si assume $T_1(N)$ uguale a al tempo che si ottiene moltiplicando per p il tempo necessario per risolvere su 1 processore il problema di dimensione $N_{loc} = \frac{N}{p}$:

$$T_1(N) \simeq p \times T_1(N_{loc})$$

Si ha cioè in questo caso la definizione operativa per il calcolo dello speed up:

$$SS_p = \frac{pT_1(N_{loc})}{T_p(pN_{loc})}$$

Alla quantitá SS_p si dá il nome di speed up **scalato**, volendo evidenziare che esso é calcolato variando sia p che N, ovvero *scalando* il problema.

♣ Esempio 21

Consideriamo la somma di $N = np$ numeri su p processori. Sia $N = 32$ e $p = 4$. Calcoliamo

$$S_p = \frac{T_1(np)}{T_p(np)}$$

Si ha:

$$T_1(np) = np - 1 = 31$$
$$T_1(n) \times p = (n-1) \times p = 7 \times 4 = 28$$

e avendo posto

$$T_1(np) \simeq p \times T_1(n) \quad \Longrightarrow \quad T_1(32) = 31 \cong 4 \cdot T_1(8) = 4 \times 7 = 28$$

segue

$$SS_p = \frac{T_1(32)}{T_4(32)} \cong \frac{4T_1(8)}{T_4(4 \cdot 8)} = \frac{28}{9} = 3.11$$

△

Se si rapporta lo speed-up scalato al numero di processori p si ottiene l'*efficienza* scalata

$$ES_p = \frac{SS_p(n)}{p} = \frac{pT_1(n)}{pT_p(pn)} = \frac{T_1(n)}{T_p(pn)}$$

É possibile ottenere speed up (scalati) prossimi allo speed up ideale. Nasce peró l'esigenza di elaborare un nuovo modello alla base del quale ci sia l'assunzione che la complessitá computazionale del problema debba aumentare al crescere della potenza computazionale a disposizione[5].

Nel 1988 Gustafson ha proposto un nuovo modello per la stima delle prestazioni di un algoritmo parallelo [29]. Alla base del nuovo modello c'è la considerazione che, avendo a disposizione elevate potenze di calcolo, la complessità computazionale dell'algoritmo sul singolo processore, "*scala*" con la potenza di calcolo. La quantità fissa non è la dimensione del problema, come nella formulazione classica dello speed up, ma piuttosto il tempo che è necessario alla risoluzione del problema (*fixed - time model*)[6]. L'idea alla base del modello di Gustavson é quella di misurare lo speed up valutando, a partire dal tempo richiesto su p processori, quanto tempo sarebbe stato necessario a eseguirlo su un solo processore. In altre parole, si misura T_1 dato T_p.

Detta $\alpha^{'}$ la frazione del tempo T_p relativa alle operazioni eseguite in sequenziale e $(1-\alpha^{'})$ la frazione del tempo T_p relativa alle operazioni

[5] Si osservi che avendo trattato come caso di studio il calcolo della della somma di n numeri che ha una complessità computazionale lineare rispetto alla dimensione del problema, abbiamo fino ad ora assimilato la complessità di tempo $T_1(n) = n - 1$ con la dimensione del problema, n.

[6] Poiché il modello dovuto ad Amadhal è stato sviluppato mantenendo fissa la dimensione del problema, lo speed up classico è detto anche *fixed-size model*.

eseguite in parallelo, calcoliamo T_1. Nel calcolo di T_1 a partire da T_p, oltre al contributo relativo alla parte sequenziale, bisogna aggiungere la parte parallela moltiplicata per p, numero dei processori, visto che stiamo misurando il tempo di esecuzione su un solo processore. Si ha quindi:

$$T_1 = \alpha^{'} T_p + p(1 - \alpha^{'}) T_p$$

da cui:

$$SS_p = \frac{T_1}{T_p} = \frac{\alpha^{'} T_p + p(1 - \alpha^{'}) T_p}{T_p} = \alpha^{'} + p(1 - \alpha^{'}) = p + (1 - p)\alpha'$$

pertanto sussiste la seguente:

Definizione: *si definisce speed up scalato la quantità:*

$$SS_p = p + (1 - p)\alpha'$$

Tale modello di speed-up è dovuto a Gustafson [28, 29].

Il grafico dello speed up scalato in funzione di $\alpha^{'}$, mostrato in Fig. 4.10, è una retta con pendenza $1 - p$. Quando lo speed-up viene misurato scalando la dimensione del problema, la frazione $\alpha^{'}$ tende a zero all'aumentare del numero di processori utilizzati. È quindi possibile ottenere prestazioni elevate.

Nel misurare le prestazioni di un algoritmo parallelo al crescere del numero dei processori e della dimensione del problema si dá la seguente:

Definizione *Un algoritmo si dice scalabile se l'efficienza scalata ES_p rimane costante al crescere del numero p dei processori e della*

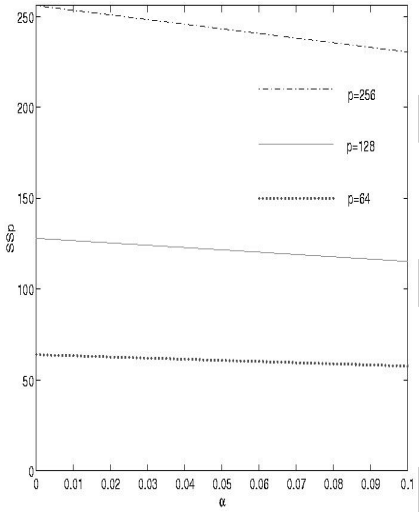

Figura 4.10: *In figura è riportato il valore dello speed up scalato in relazione ad α per diversi valori di p*

dimensione N del problema.

Al fine di progettare algoritmi scalabili nasce il seguente problema:*Fissato il numero di processori p_0, e fissata la dimensione iniziale del problema n_0 e quindi $T_1(n_0)$, calcolare n_1 e il tempo $T_1(n_1)$, affinché nel passare da p_0 a p_1 processori l'efficienza (scalata) rimanga costante. In particolare:*

$$E_{p_0}(n_0) = E_{p_1}(n_1) \tag{4.6}$$

♣ Esempio 22

Consideriamo la somma di n numeri su p processori.
Ricordiamo che, per la (4.5) e la (24):

$$E_p(n) = \frac{1}{\frac{O_h}{T_1} + 1} = \frac{1}{\frac{1-p+plog_2 p}{n-1} + 1}$$

Richiedere che:

$$E_{p_0}(n_0) = E_{p_1}(n_1)$$

equivale ad imporre che:

$$\frac{1 - p_0 + p_0 log_2 p_0}{n_0 - 1} = \frac{1 - p_1 + p_1 log_2 p_1}{n_1 - 1}$$

da cui:

$$n_1 = n_0 \cdot \frac{p_1 \log_2 p_1}{p_0 \log_2 p_0}$$

ovvero la dimensione n_1 deve aumentare, rispetto alla dimensione iniziale n_0, di un fattore k con:

$$k = \frac{p_1 \log_2 p_1}{p_0 \log_2 p_0}$$

Si osservi che il fattore k misura l'incremento dell'overhead introdotto nell'algoritmo della somma a seguito dell' aumento da p_0 a p_1 processori. Quindi, si ottiene un algoritmo scalabile se la dimensione n aumenta in maniera proporzionale all'incremento dell'overhead.
Questa considerazione é in effetti piuttosto naturale dato che l'overhead é il principale responsabile del degrado dell'efficienza all'aumentare del numero dei processori.

△

Nasce dunque in maniera naturale la seguente

Definizione *Definiamo* ***isoefficienza*** *di un algoritmo la funzione*

$$I : (p, n) \longrightarrow T_1(n, p)$$

tale che

$$E_{p_0}(n_0) = E_{p_1}(n_1)$$

Tenendo conto della (4.5) si ha:

$$\begin{cases} E_{p_0}(n_0) = \frac{1}{\frac{O_h(n_0,p_0)}{T_1(n_0)}+1} \\ \\ E_{p_1}(n_1) = \frac{1}{\frac{O_h(n_1,p_1)}{T_1(n_1)}+1} \end{cases} \Longrightarrow \frac{O_h(n_0,p_0)}{T_1(n_0)} = \frac{O_h(n_1,p_1)}{T_1(n_1)}$$

quindi

$$T_1(n_1) = T_1(n_0) \cdot \frac{O_h(n_1,p_1)}{O_h(n_0,p_0)}$$

ovvero la funzione isoefficienza è data da:

$$I: \ (p,n) \longrightarrow T_1(n) = T_1(n_0) \cdot \frac{O_h(n_1,p_1)}{O_h(n_0,p_0)} \tag{4.7}$$

La definizione di isoefficienza esprime il fatto che nella progettazione di algoritmi scalabili bisogna tener conto che all'aumentare della dimensione del problema e del numero di processori, la complessitá computazionale dell'algoritmo sequenziale debba scalare rispetto a quella iniziale in maniera proporzionale al corrispondente aumento dell'overhead, legato all'aumento del numero dei processori.

La valutazione dei tempi di esecuzione dell'algoritmo parallelo fino ad ora eseguita é semplificata perché non tiene conto del tempo di comunicazione fra processori, del tempo di sincronizzazione, etc.

♣ Esempio 23

Si esegua la somma di $N = 16$ numeri su di un calcolatore MIMD a memoria distribuita. È stato già osservato che:

- $p = 2 \Longrightarrow T_2 = 8\ t_{calc}$;
- $p = 4 \Longrightarrow T_4 = 5\ t_{calc}$;

- $p = 8 \Longrightarrow T_8 = 4\ t_{calc}$;

Consideriamo ora anche il tempo necessario per la comunicazione tra i processori. Denotiamo con t_{com} il tempo necessario ad effettuare la comunicazione di un dato f.p.. Dalla Fig. 4.2 si osserva che se $p = 2$ il tempo di comunicazione è $1\ t_{com}$. Dalla Fig. 4.3 si osserva che se $p = 4$ il tempo di comunicazione è $2\ t_{com}$. Dalla Fig. 4.4 si osserva che se $p = 8$ il tempo di comunicazione è $3\ t_{com}$.

In definitiva, se si tiene conto sia del tempo di calcolo che del tempo di comunicazione si hanno i risultati mostrati in Tabella 4.7.

p	$\mathbf{T_p}$
2	$T_2 = 8\ t_{calc} + 1\ t_{com}$
4	$T_4 = 5\ t_{calc} + 2\ t_{com}$
8	$T_8 = 4\ t_{calc} + 3\ t_{com}$

Tabella 4.7: *Nelle colonne sono indicati p, il numero di processori utilizzati, e T_p, il tempo di esecuzione, visto come somma del tempo di calcolo e del tempo di comunicazione.*

Osserviamo che nell'algoritmo della somma di n numeri su p processori, in generale si ha:

$$t_{com} = log_2 p$$

da cui:

$$T_p = \left(\frac{n}{p} - 1\right) t_{calc} + log_2 p(t_{com} + t_{calc})$$

Assumiamo, ad esempio, che sia:

$$\frac{t_{com}}{t_{calc}} = 2\ ,$$

Se si considera o meno il tempo necessario alla comunicazione, essendo

$$T_1 = 15\ t_{calc}$$

si ottengono i valori seguenti di T_p, S_p ed E_p:

Si osserva dunque che considerando il tempo di comunicazione le prestazioni dell'algoritmo possono cambiare notevolmente.

△

p	T_p	S_p	E_p
2	$8\ t_{calc}$	1.88	0.94
4	$5\ t_{calc}$	3.00	0.75
8	$4\ t_{calc}$	3.75	0.47

p	T_p	S_p	E_p
2	$10\ t_{calc}$	1.50	0.75
4	$9\ t_{calc}$	1.67	0.42
8	$10\ t_{calc}$	1.50	0.19

Tabella 4.8: *Nelle tabelle sono indicati p, il numero di processori, T_p, il tempo di esecuzione, S_p, lo speed up, ed E_p, l'efficienza. Nella prima tabella T_p coincide con il tempo di calcolo; nella seconda T_p rappresenta la somma del tempo di calcolo e del tempo di comunicazione.*

Definizione: *si definisce overhead di comunicazione unitario la quantità:*

$$OC = \frac{t_{com}}{t_{calc}}$$

L'overhead unitario è un parametro dipendente dall'ambiente computazionale (caratteristiche del processore, software di comunicazione,...). In generale, $OC > 1$. Indicato con T_p^{com} il tempo di comunicazione dell'algoritmo su p processori e con T_p^{calc} il tempo di calcolo dell'algoritmo su p processori, la quantità

$$OC_p = \frac{T_p^{com}}{T_p^{calc}} = \frac{n \cdot t_{com}}{m \cdot t_{calc}} = \frac{n \cdot OC \cdot t_{calc}}{m \cdot t_{calc}} = \frac{n \cdot OC}{m}$$

è detta *overhead di comunicazione* e fornisce una misura del *"peso"* della comunicazione sul tempo di esecuzione dell'algoritmo.

♣ Esempio 24

Tornando all'**Esempio 23** della somma di $N = 16$ numeri, supposto $OC = 2$, si ha:

Si osserva che su $p = 8$ processori il tempo di comunicazione pesa di piú rispetto al tempo di esecuzione. Tale risultato era, d'altra parte, atteso. Fissata, infatti, la

p	T_p^{calc}	T_p^{com}	OC_p
2	8 t_{calc}	1 t_{com}	0.25
4	5 t_{calc}	2 t_{com}	0.80
8	4 t_{calc}	3 t_{com}	1.50

Tabella 4.9: *Nelle colonne sono indicati p, il numero di processori, T_p^{calc}, il tempo di calcolo, T_p^{com}, il tempo di comunicazione, e OC_p, l'overhead di comunicazione.*

dimensione N del problema, all'aumentare del numero di processori aumenta il numero di comunicazioni da effettuare.

△

Oltre al tempo di comunicazione bisogna tener conto del **tempo di sincronizzazione**, ovvero il tempo di attesa necessario per coordinare il lavoro dei vari processi in esecuzione. Ad esempio, nel caso della somma di n numeri il processore P_0 deve attendere i dati dagli altri processori per poter effettuare la somma. È importante quindi che le operazioni di spedizione e ricezione di messaggi siano sincronizzate al fine di evitare inutili tempi di attesa (*idle time*) e quindi di inutilizzo dei processori. A tale proposito è importante che il carico computazionale sia **bilanciato**: i dati devono essere suddivisi tra i processori in modo tale che ognuno di essi impieghi approssimativamente lo stesso tempo per l'effettuazione del calcolo. Una distribuzione non equa del carico computazionale potrebbe far sì che alcuni processori rimangano inattivi mentre altri devono ancora effettuare numerose istruzioni, riducendo cosí l'efficienza dell'algoritmo.

♣ Esempio 25

Supponiamo di effettuare la somma di $N = 16$ numeri su due processori assegnando al primo processore 12 numeri e al secondo soltanto 4. Il primo passo è il calcolo delle somme parziali: il primo processore impiega 11 t_{calc} per sommare i 12 numeri, il secondo processore impiega 3 t_{calc} per sommare 4 numeri. Questo vuol dire che il

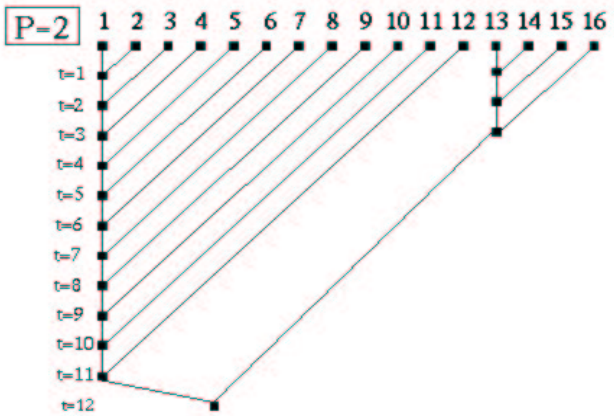

Figura 4.11: *Se il carico computazionale non è bilanciato tra i processori può accadere che alcuni di questi rimangano inattivi.*

secondo processore rimane inattivo per un tempo uguale a

$$11\ t_{calc} - 3\ t_{calc} = 8\ t_{calc}$$

in attesa che il primo processore completi la somma parziale, come mostrato in Fig. 4.11.

Calcoliamo speed-up ed efficienza. Il tempo necessario a calcolare la somma totale su di 1 processore è 15 t_{calc}. Con i dati cosí distribuiti vengono eseguite in parallelo 6 somme su 15 ed in sequenziale 9 somme su 15, 8 dovute ai numeri in piú che il processore P_0 ha avuto assegnati e 1 per sommare le due somme parziali. Risulta quindi :

$$T_2 = 12\ t_{calc}$$

effettuando 3 somme in parallelo e 9 in sequenziale,

$$S_2 = \frac{15}{12} = 1,25$$

$$E_2 = \frac{1,25}{2} = 0,62$$

Effettuando la somma di $N = 16$ numeri con 2 processori e distribuzione bilanciata, si ha (si veda Esempio 12):

$$T_2 = 8\ t_{calc}$$

e quindi

$$S_2 = \frac{15}{8} = 1,875$$

$$E_2 = \frac{1,875}{2} = 0,9375$$

Nel caso di distribuzione bilanciata si osserva, quindi, un miglioramento sia dello speed-up che dell'efficienza.

△

♣ Esempio 26

Consideriamo l'algoritmo parallelo sviluppato nel paragrafo 3.4 che esegue il prodotto matrice per matrice

$$A \cdot B = C$$

secondo lo schema BMR (V Strategia).

Supponiamo che le matrici A, B e $C \in \Re^{N \times N}$ e che i P processori siano distribuiti su di una griglia quadrata di dimensione $q \times q$. Ogni processore ha dunque i blocchi A_{loc}, B_{loc} e C_{loc} di dimensione $nb \times nb$ con[7]

$$nb = \frac{N}{q} \tag{4.8}$$

Supponiamo inoltre che:

- il tempo di invio di n dati f.p. da un processo ad un altro sia

  $$\alpha + n \cdot t_{com}$$

  dove α è il tempo di latenza e t_{com} è il tempo necessario a comunicare un dato f.p..

Calcoliamo T_P procedendo secondo lo schema dell'algoritmo BMR. Per la spedizione in broadcast[8] del blocco A_{loc} lungo le righe della griglia, da parte dei processori appartenenti alla diagonale principale, abbiamo:

$$T^1_{com} = log_2(q) \cdot (\alpha + nb^2 \cdot t_{com})$$

[7] Per semplicità di calcoli si è considerato N sia multiplo intero di q.

[8] Per effettuare il broadcast ad albero di un dato f.p., con P processori, occorrono $log_2(P)$ passi.

e per il primo prodotto parziale

$$A_{loc} \cdot B_{loc} = C_{loc}$$

abbiamo

$$T^1_{calc} = nb^3 \cdot t_{calc}$$

Ad ogni passo $k = 1, \ldots, q-1$

- I processori della $k-esima$ diagonale della griglia effettuano il broadcast di A_{loc} lungo le righe della griglia:

$$T^2_{com} = \log_2(q) \cdot (\alpha + nb^2 \cdot t_{com})$$

- Nei sottocontesti colonna della griglia, ogni processore spedisce al prorpio nord il blocco B_{loc}:

$$T^3_{com} = \alpha + nb^2 \cdot t_{com}$$

- Ogni processore calcola il prodotto parziale

$$A_{loc} \cdot B_{loc} = C_{loc}$$

per cui

$$T^2_{calc} = nb^3 \cdot t_{calc}$$

endfor

Sommando tutti i contributi relativi alle spedizioni ed alle operazioni f.p. effettuate otteniamo:

$$T_P = (\alpha + nb^2 \cdot t_{com})(q + q \cdot \log_2(q) - 1) + q \cdot nb^3 \cdot t_{calc}$$

da cui

$$E = \frac{T_1}{P \cdot T_P} = \frac{N^3\, t_{calc}}{P \cdot T_P} = \frac{N^3 t_{calc}}{(\alpha + nb^2 \cdot t_{com})(q + q \cdot log_2(q) - 1) \cdot q^2 + q^3 \cdot nb^3 \cdot t_{calc}}$$

e sostituendo la (4.8) otteniamo

$$E = \left[1 + \left(\frac{q}{N}\right)^3 \left(1 + log_2(q) - \frac{1}{q}\right) \frac{\alpha}{t_{calc}} + \frac{q}{N} \left(1 + log_2(q) - \frac{1}{q}\right) \frac{t_{com}}{t_{calc}}\right]^{-1}$$
$$= \left[1 + \left(\frac{q}{N}\right)^3 (c_1 + c_2 \cdot log_2(q)) \frac{\alpha}{t_{calc}} + \frac{q}{N} \cdot (c_1 + c_2 \cdot log_2(q)) \frac{t_{com}}{t_{calc}}\right]^{-1}$$

Si deduce quindi che al crescere di N, affinché l'efficienza E non vada a zero, il rapporto

$$\frac{q = \sqrt{p}}{N}$$

deve mantenersi costante, ovvero che sia

$$\lim_{N \to \infty} \frac{\sqrt{p}}{N} = cost$$

ciò significa che $q = \sqrt{p}$ deve crescere linearmente con N.
Volendo calcolare l'isoefficienza, risulta che

$$E(N_1, p_1) = E(N_2, p_2) \Longleftrightarrow \frac{\sqrt{p_1}}{N_1} = \frac{\sqrt{p_2}}{N_2}$$

ovvero che nel passare da p_1 a p_2 processori la dimensione del problema deve aumentare in modo tale che:

$$N_2 = N_1 \cdot \frac{\sqrt{p_2}}{\sqrt{p_1}}$$

△

In conclusione, nella valutazione delle prestazioni di un algoritmo in un ambiente di calcolo parallelo bisogna tenere conto di molti fattori tra cui:

- numero di processori;
- tempo di calcolo;
- tempo di comunicazione;
- tempo di sincronizzazione dei processori;
- bilanciamento del carico computazionale.

Questo rende necessario l'utilizzo di piú parametri (T_p, S_p, E_p, Oh_p OC_p).

Tali fattori dipendono oltre che dall'algoritmo, dall'ambiente di sviluppo, in particolare da:

- architettura e potenza dei processori;
- tipo e numero di memorie;
- connessione tra i processori;
- connessioni tra processori e memorie;
- software per il message passing.

In questo capitolo abbiamo fin qui potuto osservare che il tempo T_1 di esecuzione dell'algoritmo sequenziale gioca un ruolo fondamentale nella valutazione delle prestazioni di un algoritmo parallelo. Bisogna chiarire quindi quale sia l'algoritmo sequenziale di riferimento con cui si misura T_1. Le scelte possibili sono due:

1. T_1 è il tempo di esecuzione del *"miglior algoritmo"* sequenziale;
2. T_1 è il tempo di esecuzione dell'algoritmo parallelo su un solo processore.

Se si effettua la prima scelta lo speed up fornisce informazioni sul guadagno che si ottiene risolvendo un problema con p processori anziché con uno solo, ed è questo che interessa se si pone come obiettivo finale la risoluzione del problema nel minor tempo possibile. Purtroppo tale scelta comporta alcune difficoltà: spesso non è possibile stabilire quale sia il miglior algoritmo sequenziale.

Se si effettua la seconda scelta lo speed up dà informazioni su quanto un algoritmo sia adatto all'implementazione su una data architettura parallela.

La scelta di T_1, ovvero dell'algoritmo sequenziale di riferimento, non è in generale un problema semplice, e richiede di volta in volta un'analisi attenta dell'algoritmo e delle risorse hardware/software a disposizione.

... The mission of numerical Analysis is
to provide the scientific community
with effective software tools ...
G. Golub, 1989

... La missione dell'Analisi Numerica è
fornire alla comunità scientifica
strumenti software effettivi ...