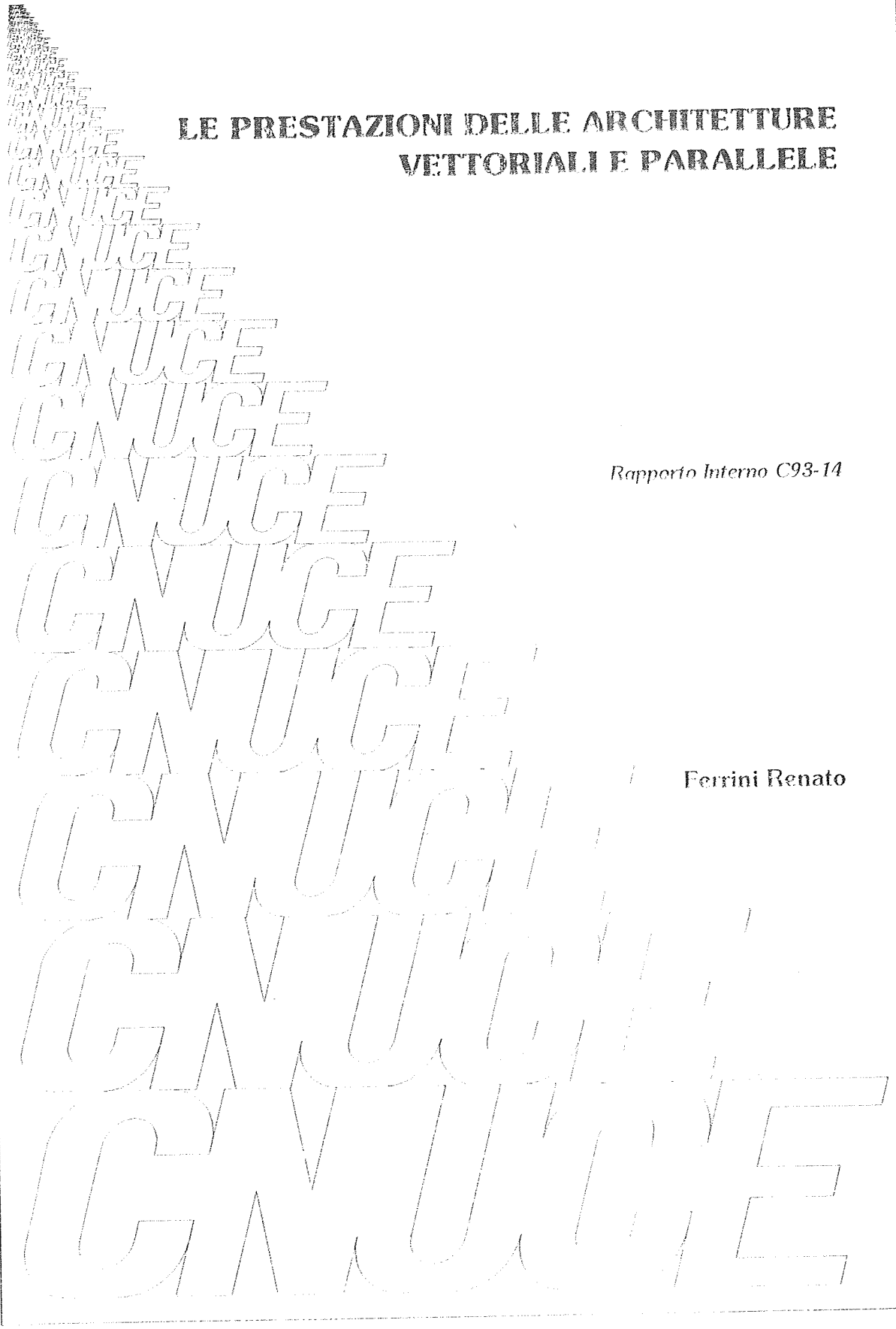
LE PRESTAZIONI DELLE ARCHITETTURE VETTORIALI E PARALLELE



Rapporto Interno C93-14

Ferrini Renato

Le prestazioni delle architetture vettoriali e parallele

Renato Ferrini

Rapporto Interno C93-14

Pisa, Aprile 1993

Indice

Introduzione

La finalità principale delle architetture vettoriali, come del resto di tutte le architetture parallele, è quella di rendere più veloce l'esecuzione di un programma; in questo contesto le prestazioni diventano uno degli elementi caratterizzanti di un elaboratore e di conseguenza i criteri di valutazione vengono ad assumere una notevole importanza. Infatti conoscere qual è la massima capacità computazionale che una applicazione con determinate caratteristiche può ottenere su di una particolare architettura vettoriale costituisce uno degli aspetti più interessanti per un utente che viene per la prima volta a contatto con macchine ad elevata potenza.

Per tale motivo le prestazioni vettoriali meritano un ampio spazio ed sono state l'elemento cruciale che ha portato alla scrittura di questa nota interna. L'argomento sarà trattato sia dal punto di vista teorico, con la definizione della potenza massima di una macchina vettoriale, che pratico, fornendo gli strumenti che sono a disposizione di un programmatore per valutare le prestazioni del proprio programma.

La potenza degli elaboratori

La potenza di picco

Il primo elemento significativo di un elaboratore vettoriale, o parallelo, è dato dalla potenza dell'architettura sia scalare che vettoriale, cioè dalla quantità di *lavoro* che un elaboratore è in grado di compiere in un determinato intervallo di tempo. Nell'ambito del calcolo vettoriale e parallelo vengono considerate, come lavoro, solo le istruzioni che producono un risultato *utile* e quindi solo quelle che effettuano delle operazioni aritmetiche su numeri in virgola mobile (*floating point)* rappresentati sia in singola che in doppia precisione. Pertanto la potenza ***p*** di una macchina vettoriale è definita dall'espressione *1* dove il tempo di esecuzione è espresso in secondi:

$$p = \frac{\text{numero di operazioni in virgola mobile}}{\text{tempo di esecuzione}} \qquad (1)$$

La relazione *1* rappresenta i ***FLOPS*** (*FLoating point OPerations per Second*) o *FLOP/S*, cioè operazioni in virgola mobile al secondo. Però l'unità di misura più frequentemente usata è il ***MFLOPS*** (*Million of FLoating point OPerations per Second*), che rappresenta i milioni di operazioni in virgola mobile eseguite in un secondo:

$$\overline{p} = \frac{p}{10^6} \qquad (2)$$

Poichè ogni tipo di operazione aritmetica richiede un proprio numero di cicli di macchina, è ovvio che nello stesso intervallo di tempo ne possono essere eseguite un numero diverso e quindi la potenza $\overline{p}$ varia in funzione dell'operazione svolta. Si hanno pertanto delle potenze più alte in presenza di operazioni

semplici come la somma e la sottrazione, che necessitano di pochi cicli di macchina, e potenze inferiori nel caso della moltiplicazione e divisione. Pertanto a seconda della frequenza dei vari tipi di operazioni all'interno di un programma si ottengono delle prestazioni differenti.

Un altro elemento che influisce sul tempo di esecuzione, e quindi sulla potenza di un elaboratore, è rappresentato dai ritardi dovuti all'accesso agli operandi residenti in memoria centrale ed alla memorizzazione del risultato. Gli effetti causati dal trasferimento dei dati da e verso la memoria sono difficilmente quantificabili, in quanto dipendono sia dall'architettura che dalla situazione momentanea in cui viene eseguita l'operazione (uno dei due operandi si trova già in un registro oppure entrambi gli operandi risiedono in una memoria secondaria, ecc...). Per questo motivo viene generalmente fornita la **potenza di picco** (*peak performance*) di una macchina vettoriale che rappresenta la potenza espressa a regime e cioè tenendo conto solamente del tempo di esecuzione richiesto dal processore.

La potenza di picco scalare e vettoriale

Andiamo adesso a definire la potenza di picco di una architettura scalare e per raggiungere tale finalità è necessario calcolare il tempo necessario per l'esecuzione seriale di una operazione aritmetica tra due numeri. Convenzionalmente esprimeremo il suddetto tempo in funzione del ciclo di macchina che sarà indicato con il simbolo τ. Sempre nell'ipotesi che i dati siano già disponibili all'unità di esecuzione, il calcolo tra due operandi richiede solo i cicli di macchina necessari allo svolgimento della operazione aritmetica, che supponiamo essere uguali a *m*. Quindi la potenza di picco di una macchina scalare è espressa da:

$$\overline{P}_{(s)} = \frac{1}{m\ \tau\ 10^6} \qquad (3)$$

Per quanto riguarda una architettura vettoriale il tempo di esecuzione di una operazione aritmetica è, a regime, uguale a τ in quanto, dopo la fase di preparazione della *pipeline*, si ha un risultato ad ogni ciclo di macchina. Pertanto la potenza di picco che si ottiene eseguendo la stessa operazione su tutti gli elementi di due vettori è espressa da:

$$\overline{p} = \frac{1}{\tau \ 10^6} \qquad (4)$$

Per sua definizione la potenza di picco rappresenta il numero massimo di operazioni che possono essere eseguite da una certa architettura e quindi, nel caso di elaboratori vettoriali, indica il *limite* al quale tendono le prestazioni quando i tempi di preparazione della *pipeline* si approssimano allo zero. In un ambiente vettoriale questa condizione si verifica man mano che le dimensioni dei vettori sono sempre più grandi; si può pertanto pensare che, in teoria, le prestazioni pari alla potenza di picco si raggiungano con l'elaborazione di vettori aventi *infiniti* elementi. Per tale motivo la potenza di picco delle macchine vettoriali spesso è rappresentata come:

$$p_\infty = \frac{1}{\tau} \text{ FLOPS} \qquad \overline{p}_\infty = \frac{1}{\tau \ 10^6} \text{ MFLOPS} \qquad (5)$$

La potenza vettoriale

Nella realtà le prestazioni sono molto diverse, in quanto è vero che le modalità di esecuzione vettoriale prevedono a regime lo svolgimento di una operazione aritmetica ad ogni ciclo di macchina, ma questa condizione si verifica dopo che la *pipeline* è stata opportunamente preparata. Ciò significa che occorrono un certo numero di cicli di macchina, qui convenzionalmente indicati con il simbolo *c*, per mettere in grado la *pipeline* di lavorare a pieno ritmo e dovuti essenzialmente all'accesso degli operandi in memoria centrale ed alla memorizzazione del risultato. Inoltre, se *m* rappresenta il numero di passi elementari, o stadi, necessari per effettuare l'operazione aritmetica ed ognuno dei quali richiede un ciclo di macchina, allora il primo risultato si ottiene in un tempo pari a $m\ \tau$.

Pertanto il calcolo tra i primi due elementi dei vettori necessita di un tempo dato da:

$$t_1 = c\ \tau + m\ \tau \qquad (6)$$

Se i vettori da elaborare hanno dimensione pari a *n*, dopo il tempo della *6*, il calcolo dei restanti *n-1* elementi avviene con una durata pari a $(n-1)\ \tau$, poichè ad ogni ciclo di macchina viene prodotto un risultato dell'operazione. Quindi il tempo vettoriale

per l'elaborazione di tutti gli elementi dei vettori è dato dalla relazione *7*.

$$t_n = t_1 + (n - 1)\,\tau = (c + m + n - 1)\,\tau \qquad (7)$$

Il tempo della *7* può essere riscritto come:

$$t_n = n\,\tau + (c + m - 1)\,\tau \qquad (8)$$

Se si assume convenzionalmente che l'elaborazione vettoriale di *n* elementi richieda un tempo di $n\ \tau$, la nuova forma del tempo di esecuzione data dalla *8* evidenzia che la *preparazione* della *pipeline* necessita di un tempo pari a $(c + m - 1)\ \tau$. Poichè questo tempo rimane costante al variare della dimensione dei vettori che devono essere elaborati, ne consegue che la sua incidenza sul tempo totale di esecuzione diminuisce al crescere di *n*.

Le valutazioni di Hockney

Una domanda che viene a questo punto spontanea è di chiederci qual è il numero di elementi dei due vettori che devono essere elaborati prima di avere un sensibile aumento delle prestazioni.

Il grafico di figura *13* rappresenta l'andamento lineare del tempo dell'esecuzione vettoriale t_n al variare del numero di elementi *n* dei vettori, come si ricava dalla relazione *8*. Si può notare che la retta intercetta l'asse dei tempi in un punto diverso dall'origine degli assi e ciò a riprova del fatto che in teoria l'elaborazione di un vettore con zero elementi richiederebbe già dei cicli di macchina.

Se esprimiamo la relazione del tempo di esecuzione in funzione dei valori *a* e *b* che rappresentano rispettivamente le intersezioni della retta con l'asse *n* e l'asse *t*, otteniamo una equazione del tipo:

$$t_n = \frac{b}{a}\,n + b \qquad (9)$$

Confrontando ora i coefficienti presenti nell'espressione *9* con quelli della *8* si ricavano le seguenti relazioni:

$$\frac{b}{a} = \tau \qquad (10)$$

$$b = (c + m - 1)\ \tau \qquad (11)$$

Come era prevedibile, l'intersezione *b* della retta t_n con l'asse dei tempi rappresenta il tempo di inizializzazione della *pipeline*. Vediamo adesso di ricavare il valore di *a*, che può essere ottenuto sostituendo l'espressione di *b* della *11* nella *10* e da cui si ottiene:

$$a = c + m - 1 \qquad (12)$$

La relazione *12* indica che il valore di *a* è uguale al numero di cicli che vengono spesi nell'inizializzazione della *pipeline*.

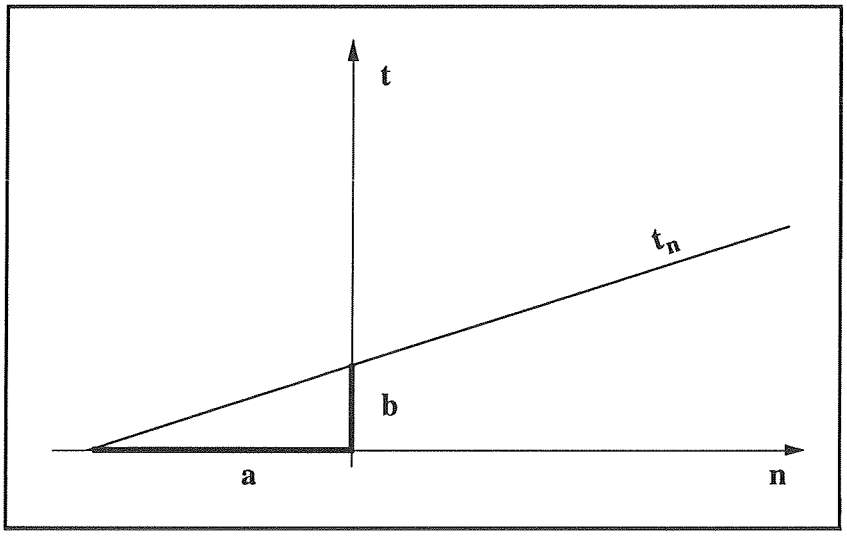


Figura 13 - Andamento del tempo dell'esecuzione vettoriale

L'interpretazione che deve essere data al parametro *a* è quella di una *dimensione negativa* del vettore che denota la presenza di elementi *fittizi* da elaborare in aggiunta agli *n* elementi reali. Infatti, a riprova di quanto detto, il tempo di esecuzione della *8* potrebbe essere riscritto come:

$$t_n = \tau\ n + \tau\ a = \tau\ (n + a) \qquad (14)$$

La relazione *14* evidenzia l'importanza del parametro *a*, in quanto fornisce un altro elemento caratterizzante di una macchina vettoriale e cioè quanto tempo di esecuzione deve essere speso prima di iniziare l'elaborazione degli elementi dei vettori. Per questo motivo cerchiamo di fornire un'altra definizione di questo parametro.

La nuova indagine è volta a valutare l'andamento delle prestazioni al variare della lunghezza dei vettori elaborati. Per definizione le prestazioni ottenute con l'elaborazione di due vettori di n elementi sono date da:

$$\overline{p}_n = \frac{n}{t_n \, 10^6} \tag{15}$$

La relazione *15* è facilmente intuibile poichè l'elaborazione tra due vettori di n elementi prevede n operazioni in virgola mobile che richiedono un tempo di esecuzione che abbiamo assunto essere pari a t_n. Se tale tempo è espresso mediante la *9* le prestazioni diventano:

$$\overline{p}_n = \frac{n}{(\frac{b}{a} \, n + b) \, 10^6} \tag{16}$$

da cui si può dedurre un primo limite della curva, e cioè quando n vale 0 le prestazioni sono nulle:

$$\overline{p}_0 = \frac{0}{b \, 10^6} = 0 \tag{17}$$

Normalizzando la *16* rispetto ad n si ha:

$$\overline{p}_n = \frac{1}{(\frac{b}{a} + \frac{b}{n}) \, 10^6} \tag{18}$$

La *18* fornisce due indicazioni importanti, di cui la prima è l'andamento crescente della curva e la seconda che con n tendente all'infinito le prestazioni valgono:

$$\overline{p}_n = \frac{1}{\frac{b}{a} \, 10^6} \tag{19}$$

Mediante le relazioni *10* e *5* si ottiene:

$$\overline{p}_n = \frac{1}{\tau \, 10^6} = \overline{p}_\infty \tag{20}$$

che era il risultato aspettato. L'andamento delle prestazioni al variare della dimensione dei vettori elaborati è pertanto quello illu-

strato in figura *21*.

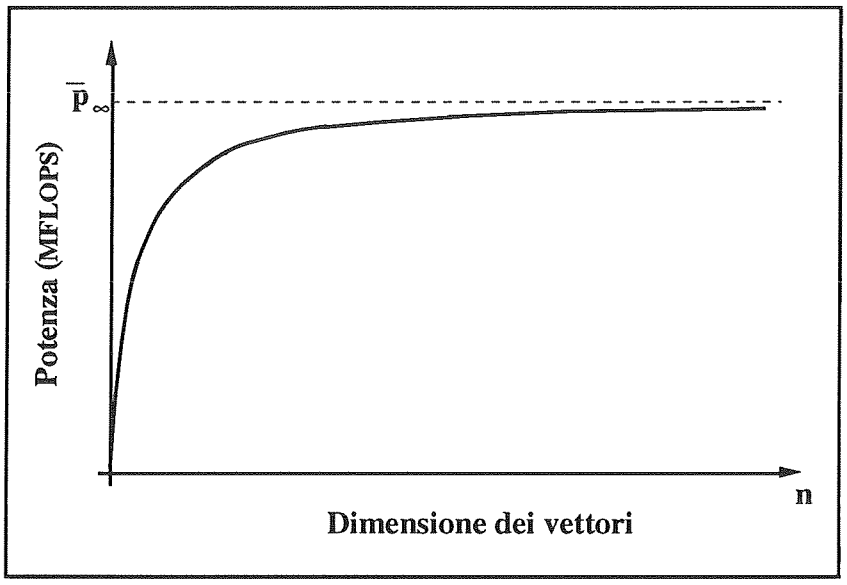


Figura 21 - Andamento delle prestazioni in funzione della dimensione dei vettori

Il fattore $n_{1/2}$

Definiamo adesso la dimensione dei vettori che è necessaria per ottenere la metà della potenza di picco ed indichiamo questo numero con $\mathbf{n_{1/2}}$ (da leggersi come *n di un mezzo* e non come *n mezzi* che generalmente indica la metà della lunghezza dei vettori di *n* elementi). La figura *25* illustra graficamente il valore di $n_{1/2}$.

Il valore delle prestazioni quando la dimensione dei vettori vale $n_{1/2}$, è dato per definizione da:

$$\overline{P}_{n_{1/2}} = \frac{\overline{P}_\infty}{2} = \frac{1}{2\,\frac{b}{a}\,10^6} = \frac{1}{(\frac{b}{a} + \frac{b}{a})\,10^6} \qquad (22)$$

mentre per la relazione *18* le prestazioni valgono:

$$\overline{P}_{n_{1/2}} = \frac{1}{(\frac{b}{a} + \frac{b}{n_{1/2}})\ 10^6} \qquad (23)$$

Dal confronto della *22* con la *23* si ricava che:

$$n_{1/2} = a \qquad (24)$$

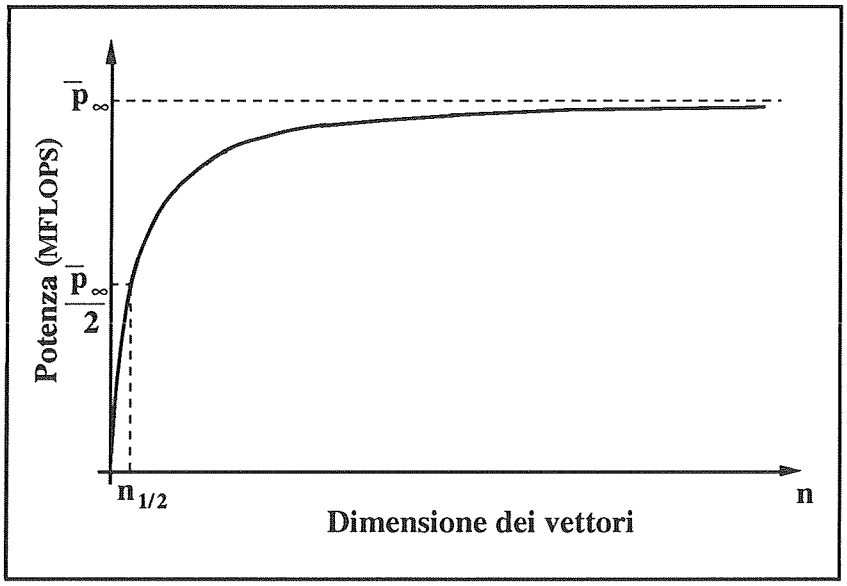


Figura 25 - Rappresentazione del valore di $n_{1/2}$

Quindi il numero degli elementi *fittizi* (variabile *a*) che devono essere elaborati per inizializzare la *pipeline* è uguale alla dimensione dei vettori con cui si ottengono delle prestazioni pari alla metà della potenza di picco della macchina vettoriale (variabile $n_{1/2}$). Questa asserzione permette di stimare per ogni architettura vettoriale il tempo di esecuzione che deve essere *perso* prima di avere un tempo *utile* di elaborazione: infatti se calcoliamo il tempo di esecuzione della *9* quando la dimensione dei vettori è pari a $n_{1/2}$ si ottiene:

$$t_{n_{1/2}} = \frac{b}{a}\ n_{1/2} + b = \frac{b}{a}\ a + b = 2\ b \qquad (26)$$

Il risultato, che è mostrato visivamente in figura *27*, evidenzia che il parametro $n_{1/2}$ non solo indica la dimensione *negativa* del vettore causata dalla fase di preparazione della *pipeline*, ma è an-

che quella lunghezza dei vettori per la quale il tempo perso nella inizializzazione della *pipeline* coincide con il tempo utile di elaborazione. In altre parole la dimensione $n_{1/2}$ stabilisce quella soglia al di sotto della quale il tempo speso nella fase di preparazione della *pipeline* prevale sul tempo di esecuzione delle operazioni vettoriali, mentre al di sopra di tale soglia il tempo di elaborazione è più grande del tempo di inizializzazione.

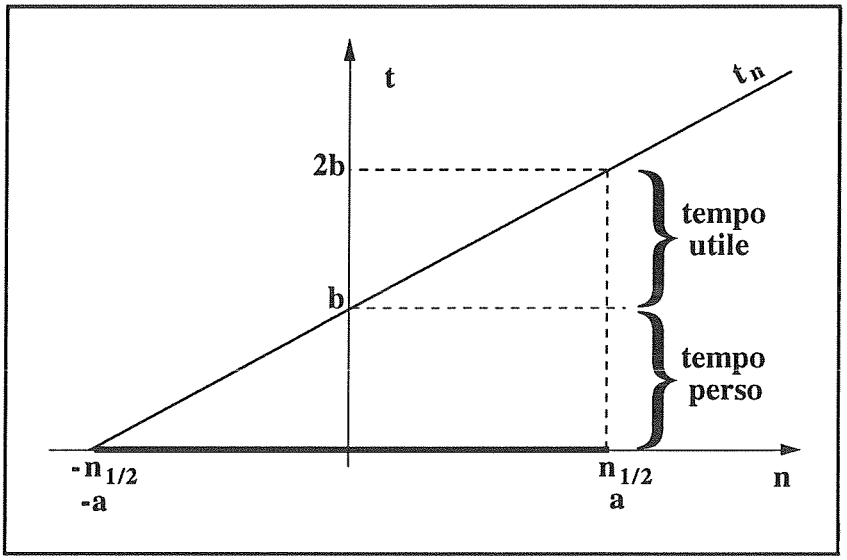


Figura 27 - Andamento del tempo di esecuzione di vettori con $n_{1/2}$ elementi

Le considerazioni appena esposte portano ad una importante conclusione: per ottenere delle buone prestazioni vettoriali è necessario che la dimensione dei vettori da elaborare sia **più grande** di $n_{1/2}$.

La determinazione di $n_{1/2}$

Data l'importanza del parametro $n_{1/2}$ la sua valutazione rappresenta un indice significativo dell'efficienza vettoriale di un elaboratore, perchè rappresenta il numero di elementi di un vettore che occorrono per superare la metà della potenza di picco e quindi iniziare ad avvicinarsi alle prestazioni ottimali. È ovvio che $n_{1/2}$ non è espresso da unico valore per una determinata architettura vettoriale; infatti dalla *24* si ricava:

$$n_{1/2} = a = c + m - 1 \qquad (28)$$

Come si può facilmente osservare il parametro $n_{1/2}$ dipende dal numero m dei cicli di macchina occorrenti per riempire la *pipeline* ed è noto che tale numero varia a seconda del tipo di operazione aritmetica da elaborare. Pertanto la soglia minima della lunghezza dei vettori, oltre la quale si iniziano ad ottenere dei significativi miglioramenti delle prestazioni, non è fissa ma si alza quanto più complesse sono le operazioni da svolgere.

Il valore corretto di $n_{1/2}$ di una macchina vettoriale può essere solamente dedotto dalle sue caratteristiche architetturali e quindi dalla esatta conoscenza dei cicli di macchina necessari alla inizializzazione della *pipeline*. Spesso, però, tali informazioni sono di difficile reperimento ed in questo caso si può ricorre ad una approssimazione empirica di $n_{1/2}$ che però ha il vantaggio di rendere sempre possibile il calcolo.

Vediamo come si ottiene questa approssimazione e per raggiungere il nostro scopo partiamo dalla relazione *9* che può essere riscritta come:

$$t_n = \frac{b}{a} n + b = \frac{b}{a} (n + a) = \tau (n + n_{1/2}) \qquad (29)$$

Dalla definizione della potenza di picco di una macchina vettoriale (prima relazione della *5*) risulta che:

$$\tau = p_\infty^{-1} \qquad (30)$$

Sostituendo la *30* nella *29* si ha:

$$t_n = p_\infty^{-1} (n + n_{1/2}) \qquad (31)$$

A questo punto è possibile ricavare $n_{1/2}$ come:

$$n_{1/2} = t_n p_\infty - n = n \left(\frac{t_n p_\infty}{n} - 1\right) \qquad (32)$$

Tenendo presente che il rapporto n/t_n rappresenta le prestazioni, espresse in *FLOPS*, dell'elaborazione di due vettori aventi n elementi, si ottiene:

$$n_{1/2} = n \left(\frac{p_\infty}{p_n} - 1\right) \qquad (33)$$

La relazione *33* indica che il valore di $n_{1/2}$ può essere ricavato dalla potenza di picco della macchina vettoriale e dalle prestazioni ottenute elaborando vettori di dimensione n. Questo risultato dimostra appunto l'approssimazione della misura di $n_{1/2}$ usando la relazione *33*, in quanto si rende necessario ricavare un parametro mediante un'esecuzione reale che introduce sempre un fattore di errore dovuto sia alla metodologia di misurazione che ad elementi di disturbo come la multiprogrammazione. Ma, anche se in una forma alterata da un fattore che normalmente si aggira intorno a qualche punto percentuale, il parametro $n_{1/2}$ rimane comunque significativo.

Nella *33* le prestazioni p_∞ e p_n sono espresse in *FLOPS*, mentre in genere si preferisce misurare le prestazioni in *MFLOPS*, e per tale motivo la *33* è più usata nella forma:

$$n_{1/2} = n \left(\frac{\overline{p}_\infty}{\overline{p}_n} - 1\right) \qquad (34)$$

Il calcolo di $n_{1/2}$

Cerchiamo adesso di calare tutte le definizioni date in un esempio pratico. Supponiamo di avere una ipotetica macchina vettoriale che ha un ciclo di macchina pari a 16 nsec; da questo primo elemento si ricava un la potenza di picco:

$$\overline{p}_\infty = \frac{1}{16\ 10^{-9}\ 10^{6}} = 62.5 \text{ MFLOPS} \qquad (35)$$

Per ottenere, nella pratica, le prestazioni raggiunte con l'elaborazione di vettori di n elementi sarebbe necessario, a questo punto, procedere ad un'esecuzione di un programma di prova e ricavare il parametro t_n. Le modalità con cui deve essere effettuata la misura saranno fornite più avanti, in questo momento basti solo sapere che esiste la possibilità di ottenere con una buona approssimazione il tempo speso nell'esecuzione di n operazioni dello stesso tipo. Ma poichè la macchina vettoriale è ipotetica ci limitiamo a fornire dei probabili tempi di esecuzione relativi alle quattro operazioni aritmetiche in modo che siano abbastanza aderenti alla realtà. Il fatto di eseguire separatamente ogni operazione aritmetica serve per evidenziare i differenti valori di $n_{1/2}$ che si ottengono in relazione all'operazione svolta.

La tabella *36* mostra appunto i risultati che sono stati dedotti con l'elaborazione di vettori lunghi 100 elementi su di una architettura con un ciclo di macchina di 16 nsec.

Gli ipotetici tempi di esecuzione presenti nella tabella consentono di risalire alle prestazioni $\overline{p}_n$ mediante la relazione *15* e da queste alla determinazione del parametro $n_{1/2}$ utilizzando l'espressione *34*.

Operazione	t_n (μsec)	$\overline{p}_n$ (MFLOPS)	$n_{1/2}$
$a_i + b_i$ $a_i - b_i$	2.1	47.6	31.3
$a_i * b_i$	2.6	38.4	62.7
a_i / b_i	3.1	32.2	94

Tabella 36 - Prestazioni delle quattro operazioni aritmetiche

I risultati mostrati sono prevedibili, in quanto le operazioni *veloci* come la somma e la sottrazione, che richiedono pochi cicli di macchina, permettono di raggiungere delle prestazioni maggiori della metà della potenza di picco con un numero minore di elementi, mentre operazioni più complesse necessitano di vettori con una dimensione maggiore.

Le prestazioni con i registri vettoriali

Tutte le valutazioni fin qui svolte sulle prestazioni degli elaboratori vettoriali sono partite dall'ipotesi che il tempo di esecuzione cresca linearmente con l'aumentare della dimensione dei vettori e quindi possa essere espresso secondo la funzione *8*. Questa condizione è soddisfatta per le macchine vettoriali in cui la *pipeline* è alimentata da dati provenienti dalla memoria centrale e dove anche i risultati sono direttamente trasferiti in memoria. In-

vece, in presenza dei registri vettoriali, subentra durante il calcolo un elemento importante: il *sezionamento*. Questo significa che il flusso dei dati verso la *pipeline* non è lineare, ma ogni z elementi (dove z rappresenta la dimensione della sezione) si verifica una specie di *interruzione* dovuta alla memorizzazione dei risultati nella memoria centrale ed al caricamento dei registri vettoriali con i nuovi dati da elaborare.

Il sezionamento provoca quindi una improvvisa impennata del tempo di esecuzione, il quale subisce successivamente un incremento modesto durante l'elaborazione di tutti gli elementi della sezione, poichè, essendo già state risolte tutte le operazioni di accesso ai dati e di inizializzazione della *pipeline*, le prestazioni sono prossime alla potenza di picco della macchina. Questo effetto è visualizzato in figura *37*.

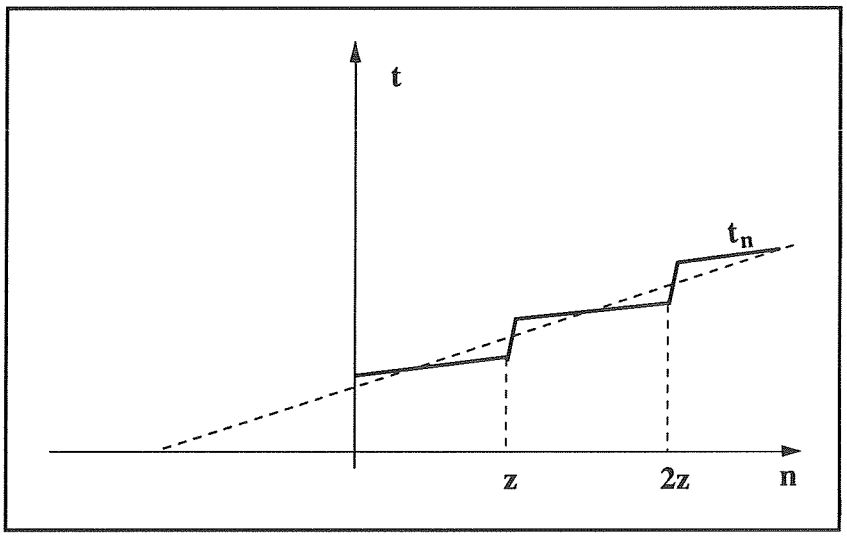


Figura 37 - Andamento del tempo di esecuzione con il sezionamento dei vettori

Il grafico mostra il considerevole incremento del tempo di esecuzione quando la dimensione dei vettori passa da z a $z+1$ elementi. Però, una volta superata la soglia z della sezione, la variazione del tempo di calcolo è modesta finchè il numero di elementi rimane all'interno della sezione e per tale motivo non è conveniente lavorare con vettori la cui lunghezza è di poco superiore alla dimensione della sezione.

Nel caso di elaborazione con registri vettoriali il valore assunto da $n_{1/2}$ non è univoco a causa delle oscillazioni del tempo di esecuzione. Si potrebbe comunque prendere in considerazione la retta tendenziale ricavata dall'andamento della curva, che nel grafico di figura *37* è rappresentata dalla linea tratteggiata, ma in

questo caso il parametro $n_{1/2}$ avrebbe solo un valore teorico e non pratico.

Anche il valore delle prestazioni di un elaboratore provvisto di registri vettoriali risente dell'influsso del sezionamento sul tempo di esecuzione, come è mostrato dall'andamento del grafico di figura *38*. Le prestazioni subiscono un deciso calo quando la dimensione dei vettori supera la soglia della sezione, ma però questa diminuzione si attenua man mano che il valore di n diventa sempre più grande.

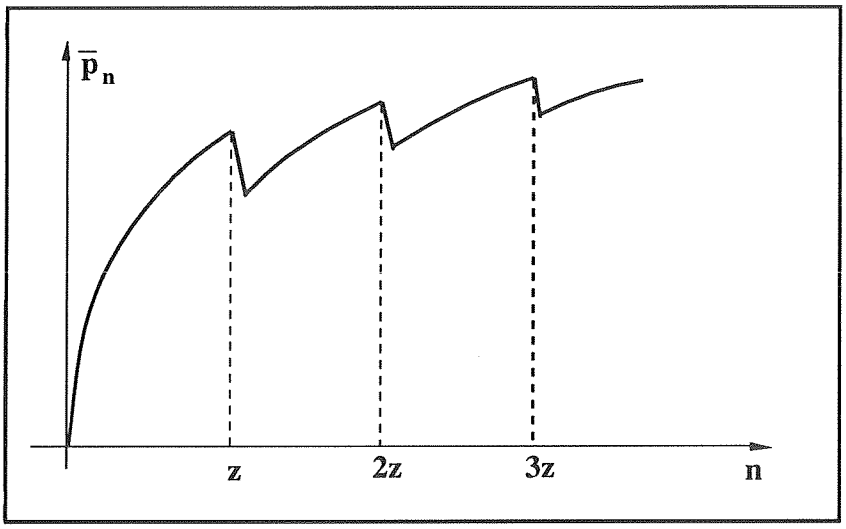


Figura 38 - Andamento delle prestazioni vettoriali con il sezionamento

Queste considerazioni sulle modalità di esecuzione degli elaboratori provvisti di registri vettoriali portano a concludere che non esiste un solo valore di n oltre il quale aumentano sensibilmente le prestazioni, come accade per le macchine *memory to memory*, ma esistono degli intervalli di valori determinati dalle seguenti condizioni:

1. *con n piccolo*
   Tutti i valori che sono lontani dalla soglia della sezione (indicativamente compresi tra $z/2$ e z).

2. *con* $n >> z$
   Tutti i valori.

Apparentemente potrebbe sembrare che la presenza dei registri vettoriali costituisca un elemento architetturale di ostacolo all'efficienza dell'esecuzione e questo è vero nel caso di operazioni singole. Se però l'istruzione da eseguire comprende, ad esempio, due somme ed una moltiplicazione, si ottengono generalmente dei maggiori vantaggi con una architettura a registri vettoriali che non con una architettura *memory to memory*, in quanto i risultati parziali delle operazioni si trovano già presenti in un registro vettoriale e quindi sono direttamente disponibili alla *pipeline*.

19

La prestazioni effettive

La misura dei tempi di esecuzione

La conoscenza delle caratteristiche di una macchina vettoriale è un fattore importante per sviluppare una efficiente programmazione. Purtroppo molte informazioni sono difficili da reperire o spesso le case costruttrici non forniscono tutti gli elementi per poter risalire al dato cercato: non resta quindi che tentare di capire il funzionamento dell'architettura mediante alcune prove pratiche. Un altro motivo per cui è frequente il ricorso alla misura dell'esecuzione è dovuto alla verifica del livello di ottimizzazione raggiunto; in altre parole, un'applicazione che non ottiene le prestazioni desiderate può essere sottoposta ad un controllo dei tempi di calcolo nelle zone di maggiore interesse per cercare di capire quali sono gli elementi che rallentano l'esecuzione.

Durante questa fase un elemento che viene ad assumere una notevole importanza è la metodologia di misura. Infatti il non tenere conto di fattori che influenzano l'esecuzione potrebbe introdurre degli errori sui tempi rilevati e quindi alterare, se non stravolgere, le conclusioni sul fenomeno da misurare. Il fatto anche di misurare la durata delle istruzioni vettoriali, che richiedono dei tempi dell'ordine dei nanosecondi, offre già un'idea dell'accuratezza con cui deve essere svolta tale operazione. Quindi, innanzitutto, è necessario usare un sottoprogramma di rilevazione dei tempi di calcolo che fornisca i dati richiesti con una precisione almeno dell'ordine del microsecondo. Normalmente questi strumenti di misura dei tempi si trovano nelle librerie di corredo delle macchine scalari e vettoriali e pertanto non è difficile individuarne uno che soddisfi l'esigenza su esposta. Da tenere presente che il tempo da misurare è il *tempo di calcolo*, cioè il tempo di lavoro del processore scalare o vettoriale, e non la durata dell'intero programma.

Per misurare correttamente le prestazioni ottenute con l'esecuzione di una operazione vettoriale sarebbe necessario in-

tervenire sul programma scritto in linguaggio macchina, perchè è l'unico modo per *isolare* le istruzioni vettoriali dalle altre istruzioni scalari. Ma poichè pochi utenti sono in grado di svolgere questo tipo di operazione, allora è opportuno agire direttamente sul programma facendo ricorso ad alcuni piccoli espedienti. Supponiamo infatti di voler conoscere il tempo di esecuzione di una somma vettoriale svolta in un programma Fortran e di disporre di un sottoprogramma di nome *TIME* che fornisce il tempo di lavoro del processore espresso in microsecondi. La rilevazione del tempo può essere ottenuta scrivendo il codice come è mostrato in figura *39*.

```
0001            CALL TIME(TEMPO1)

0002            DO  1  I=1,N
0003               A(I) = B(I) + C(I)
0004       1    CONTINUE

0005            CALL TIME(TEMPO2)
0006            TEMPO=TEMPO2-TEMPO1
0007            TMSOMMA=TEMPO/N
```

Figura 39 - Misura del tempo di esecuzione di una somma vettoriale

Poichè generalmente tutti i sistemi conservano il tempo di lavoro di un processore in un contatore che viene continuamente incrementato, è necessario effettuare la sua registrazione, mediante la chiamata al sottoprogramma *TIME*, prima e dopo l'evento da misurare e quindi ottenere il tempo richiesto come differenza dei due tempi. Nel caso del codice di figura *39* la variabile *TEMPO* contiene perciò il tempo di esecuzione del ciclo *DO* e cioè il tempo necessario a svolgere *N* somme vettoriali. Dato che la nostra finalità era quella di ottenere il tempo di esecuzione di una sola operazione, allora la divisione della variabile *TEMPO* per *N* fornisce il dato richiesto.

Sulla metodologia usata nell'esempio di figura *39* vanno però fatte alcune considerazioni. Il ciclo *DO*, oltre alle operazioni vettoriali, contiene anche alcune istruzioni scalari che servono per controllare l'iterazione dell'indice *I* e la loro esecuzione rientra nel tempo di lavoro del processore. Non solo, ma anche tutte le istruzioni del sottoprogramma *TIME*, alcune con la prima chiamata ed il resto con la seconda, vengono incluse nel tempo di esecuzione del ciclo *DO*. Se l'operazione vettoriale è molto lunga allora questi tempi aggiuntivi possono essere considerati

trascurabili, ma nel caso della somma è necessario intraprendere dei provvedimenti atti ad eliminarli.

In alcune librerie di sistema esistono dei sottoprogrammi di rilevazione dei tempi che forniscono separatamente il tempo del processore scalare da quello del processore vettoriale. In questo modo il problema si risolve automaticamente in quanto le istruzioni *spurie* vanno a far parte del tempo scalare e non incidono sul tempo da misurare che è quello vettoriale. L'unico elemento che, anche in questo caso, potrebbe continuare ad alterare la misura è costituito dalla durata del tempo di esecuzione. Infatti, poichè il sottoprogramma *TIME* fornisce i tempi in microsecondi, per non incorrere in errori dovuti a problemi di precisione, è necessario che il tempo da misurare sia almeno di un ordine di grandezza 1000 volte superiore e cioè della durata di alcuni *millisecondi*. Se ciò non avviene, allora si deve ricorrere ad un ampliamento artificioso del tempo da misurare, come è mostrato in figura *40*.

```
0001            CALL TIME(TEMPOS,TEMPOV1)

0002            DO  9  ING=1,1000
0003               DO  1  I=1,N
0004                  A(I) = B(I) + C(I)
0005      1        CONTINUE
0006      9     CONTINUE

0007            CALL TIME(TEMPOS,TEMPOV2)
0008            TEMPO=TEMPOV2-TEMPOV1
0009            TMSOMMA=TEMPO/(N*1000)
```

Figura 40 - Ampliamento del tempo di esecuzione di una somma vettoriale

Come abbiamo supposto, il sottoprogramma *TIME* in questo caso fornisce in modo separato il tempo scalare da quello vettoriale e ciò permette di rilevare con una buona precisione il tempo dovuto all'operazione vettoriale di somma, che ovviamente include anche il tempo di accesso agli operandi nella memoria centrale. L'effetto del ciclo *DO* con indice *ING* è quello di *amplificare* la durata dell'operazione facendola ripetere per un numero 1000 volte superiore rispetto al codice originale. Si ottiene così una misura più accurata del tempo cercato.

Nei casi in cui non è possibile disporre di un sottoprogramma che fornisca separatamente il tempo scalare e vettoriale, ma un unico tempo che è la somma dei due, si rende necessario il ricorso ad un altro stratagemma per eliminare gli effetti prodotti

dalla presenza delle istruzioni *spurie*. Il codice risultante diventa pertanto quello di figura *41*.

Le istruzioni *0001-0005* servono appunto per rilevare sia il tempo scalare necessario per la gestione del ciclo *DO*, che il tempo speso nell'esecuzione del sottoprogramma *TIME*. Il tempo risultante *TMCICLO* è poi detratto dal tempo totale di esecuzione dell'operazione vettoriale nell'istruzione *0011*. Questi accorgimenti consentono una buona correzione del tempo da misurare, pur non arrivando alla stessa precisione fornita dalla conoscenza diretta del tempo vettoriale.

```
0001            CALL TIME(TEMPO1)
0002            DO  9  I=1,N
0003        9   CONTINUE
0004            CALL TIME(TEMPO2)
0005            TMCICLO=TEMPO2-TEMPO1

0006            CALL TIME(TEMPO1)
0007            DO  1  I=1,N
0008               A(I) = B(I) + C(I)
0009        1   CONTINUE
0010            CALL TIME(TEMPO2)
0011            TEMPO=TEMPO2-TEMPO1-TMCICLO
0012            TMSOMMA=TEMPO/N
```

Figura 41 - Eliminazione delle istruzioni spurie dal tempo di esecuzione

Anche nel codice di figura *41* si potrebbe applicare l'effetto di amplificazione presente in figura *40*, se la dimensione del tempo di esecuzione lo rendesse necessario.

Una volta risaliti al tempo di esecuzione è possibile ricavare le prestazioni fornite dalla macchina vettoriale, mediante il rapporto:

$$\overline{P}_N = \frac{N}{\text{TEMPO } 10^6} \qquad (42)$$

dove *N* rappresenta il numero di operazioni svolte e quindi, in questo caso, il numero di somme vettoriali che sono state eseguite, e *TEMPO* è il tempo di esecuzione in secondi. Il fattore 10^6 serve per esprimere le prestazioni in termini di *MFLOPS*.

Le prestazioni scalari e vettoriali

L'analisi fin qui svolta ha avuto come obiettivo la valutazione delle prestazioni in rapporto alla potenza di picco di una macchina vettoriale, ma esiste un altro punto di vista interessante che è quello di considerare la differenza delle prestazioni tra l'esecuzione vettoriale e scalare dello stesso programma.

Se si rappresentano graficamente le due esecuzioni si ottiene una situazione come quella mostrata in figura *43*.

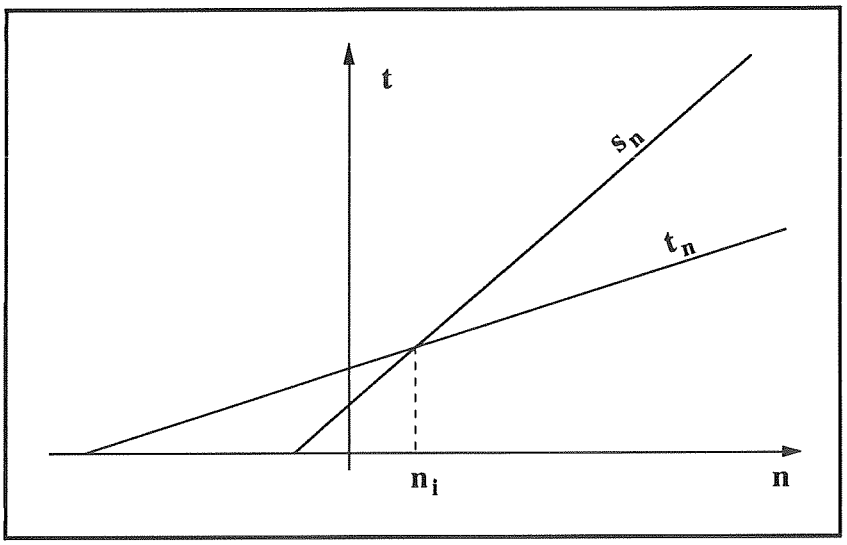


Figura 43 - Andamento dei tempi di esecuzione scalare e vettoriale

Come si può notare il tempo scalare s_n richiede un minor numero di cicli di macchina per la preparazione dell'operazione e questo è il motivo per cui la retta intercetta l'asse dei tempi in un punto più basso rispetto al tempo vettoriale. Però al crescere della lunghezza dei vettori il tempo scalare aumenta più rapidamente del tempo vettoriale e quindi esiste un punto n_i dove i due tempi coincidono. Di conseguenza è possibile trovare una dimensione dei vettori al di sotto della quale l'esecuzione scalare è **più conveniente** di quella vettoriale.

La determinazione del punto di intersezione tra le due rette può essere effettuata partendo dalle funzioni dei due tempi:

$$t_n = (c + m - 1)\,\tau + n\,\tau \qquad (44)$$

$$s_n = d\,\tau + n\,m\,\tau \qquad (45)$$

Nella *45* il tempo di preparazione dell'operazione scalare $\tau\, d$ è essenzialmente dovuto all'accesso degli operandi residenti nella memoria centrale. Eguagliando la relazione *44* con la *45* e ponendo n uguale ad n_i si ottiene:

$$d + n_i\,m = c + m - 1 + n_i \qquad (46)$$

Dalla *46* si ricava il valore di n_i:

$$n_i = \frac{c - d}{m - 1} + 1 \qquad (47)$$

La relazione *47* evidenzia come il punto di intersezione della retta del tempo scalare con quella del tempo vettoriale dipenda, oltre che dalla quantità costante *c-d*, anche, in modo inversamente proporzionale, dal numero m di cicli di macchina necessari per eseguire l'operazione aritmetica richiesta tra due operandi. Ciò significa che più l'operazione è complessa e meno elementi sono necessari affinchè l'esecuzione vettoriale sia più conveniente di quella scalare.

Quindi dal punto di vista del rapporto tra tempo vettoriale e scalare le operazioni *lente* come la divisione, che necessitano di parecchi cicli di macchina, sono quelle che offrono una maggiore efficienza di esecuzione, mentre, come abbiamo visto in precedenza, le stesse operazioni, se analizzate nell'ottica delle prestazioni assolute, sono quelle che forniscono i peggiori risultati.

Un altro problema da affrontare è di valutare il risparmio di tempo che si può ottenere passando dall'esecuzione scalare a quella vettoriale di uno stesso programma. La soluzione a questo quesito è fornita da un fattore chiamato ***speedup*** che indica quante volte l'esecuzione vettoriale è più veloce di quella scalare e che è definito come:

$$S_n = \frac{s_n}{t_n} \qquad (48)$$

Il fattore di *speedup* è quindi dato dal rapporto tra il tempo scalare e quello vettoriale e più grande risulta il suo valore e tanto maggiori sono i benefici offerti dall'esecuzione vettoriale; infatti è come se un programma, che richiede un tempo pari a s_n secondi, fosse eseguito su di una macchina S_n volte più veloce.

Anche lo *speedup* dipende dalla lunghezza dei vettori perchè, come si può facilmente dedurre dalla figura *43*, valori più grandi di n comportano un maggior divario tra il tempo scalare e vettoriale. Pertanto, se rappresentiamo graficamente l'andamento dello *speedup* al variare della dimensione dei vettori si ottiene la curva di figura *49*.

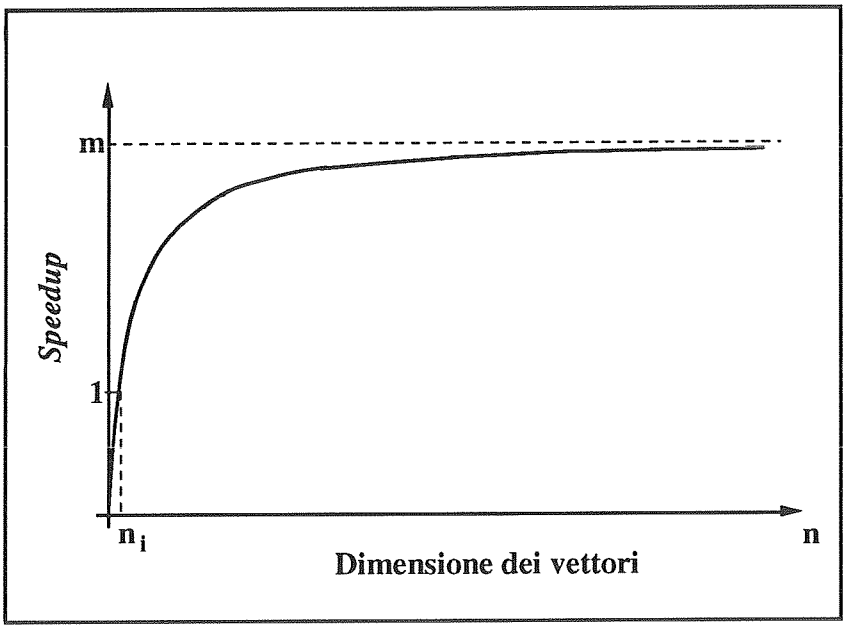


Figura 49 - Andamento dello speedup al variare della dimensione dei vettori

La curva dello *speedup* si mantiene al di sotto di 1 fino a quando la dimensione dei vettori è minore del valore n_i e tende esponenzialmente al valore limite di m, che rappresenta il numero di cicli occorrenti per l'esecuzione di una singola operazione. Questo limite si ricava considerando i tempi ottimali dell'esecuzione scalare e vettoriale per l'elaborazione di n elementi e ciò avviene quando nelle espressioni *44* e *45* sono assenti i cicli di macchina persi per la preparazione dell'operazione:

$$s_n = \tau\ n\ m \tag{50}$$

$$t_n = \tau\ n \tag{51}$$

da cui deriva lo *speedup teorico*:

$$S_n = \frac{\tau\ n\ m}{\tau\ n} = m \tag{52}$$

La *52* mostra appunto che, in assenza dei cicli di macchina richiesti dalla preparazione dell'operazione, il tempo di esecuzione scalare è m volte più grande del corrispondente tempo di esecuzione vettoriale.

La legge di Amdahl

Fino ad ora, per il calcolo dello *speedup*, sono stati presi in considerazione il tempo scalare s_n e vettoriale t_n che rappresentano i tempi richiesti per eseguire un'operazione tra due vettori di dimensione n. Nella realtà, però, un programma possiede una struttura ben più complessa del mero calcolo tra due vettori e per questo motivo è opportuno definire uno ***speedup* effettivo** che tenga conto di tutti gli elementi di programmazione presenti nelle due esecuzioni. Analogamente alla *48* anche lo *speedup effettivo* è un fattore che rappresenta il rapporto esistente tra l'esecuzione scalare e vettoriale:

$$S_{eff} = \frac{T_s}{T_v} \qquad (53)$$

Nella *53* i simboli T_s e T_v indicano rispettivamente il tempo di esecuzione scalare e vettoriale di un'applicazione. A questo proposito bisogna precisare che il confronto tra il tempo scalare e vettoriale non deve essere svolto eseguendo lo stesso programma nelle due condizioni elaborative, ma, per evitare artificiose alterazioni dello *speedup*, bisogna usare il *miglior algoritmo scalare* contro il *miglior algoritmo vettoriale*. Questa puntualizzazione si rende necessaria perchè generalmente le modalità di scrittura di un efficiente codice vettoriale sono diverse da quelle scalari e pertanto una buona applicazione vettoriale non offre delle analoghe prestazioni su di una architettura scalare. Quindi il fattore di *speedup* deve essere *depurato* da elementi che possono smiminuirne il valore intrinseco e renderlo, alla fine, poco attendibile.

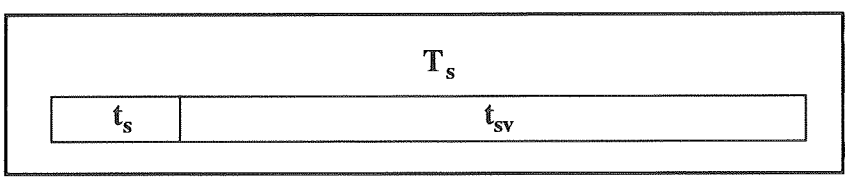


Figura 54 - Rappresentazione del tempo di esecuzione scalare

Cerchiamo adesso di analizzare i fattori che limitano lo *speedup effettivo*, considerando che i cicli di macchina persi per la preparazione delle operazioni non sono più l'unico elemento che ha un'influenza negativa sul tempo di esecuzione. Infatti da un'attenta analisi il tempo scalare può essere scomposto in due parti, come visualizzato in figura *54*, di cui t_s rappresenta la parte di codice composta dalle istruzioni non vettorizzabili, mentre t_{sv} è costituita dalla parte del codice che può essere vettorizzata.

Pertanto di tutto il tempo scalare T_s è possibile ridurre, mediante l'esecuzione vettoriale, solo la parte indicata con t_{sv}, mentre la parte t_s rimane inalterata, come si può vedere in figura *55*.

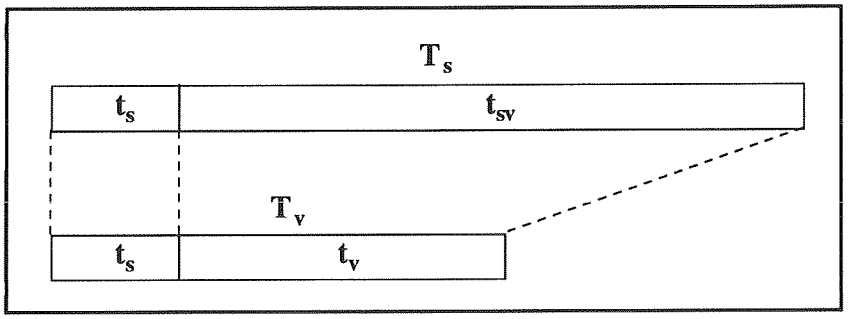


Figura 55 - Rappresentazione del tempo di esecuzione scalare e vettoriale

Come è mostrato dalla rappresentazione dei tempi, la parte di codice t_s non subisce alcuna variazione nel passaggio dall'esecuzione scalare a quella vettoriale e quindi, di fatto, costituisce un limite al valore dello *speedup*. Questo effetto risulta anche evidente se lo *speedup* è espresso sotto una forma diversa. Infatti la *53* può essere riscritta come:

$$S_{eff} = \frac{t_s + t_{sv}}{t_s + t_v} \qquad (56)$$

Normalizzando per la quantità del numeratore si ottiene:

$$S_{eff} = \frac{1}{\frac{t_s}{t_s + t_{sv}} + \frac{t_v}{t_s + t_{sv}}} = \frac{1}{(1 - \frac{t_{sv}}{t_s + t_{sv}}) + \frac{t_{sv}}{t_s + t_{sv}} \frac{t_v}{t_{sv}}} \qquad (57)$$

Considerando che il rapporto $t_{sv}/(t_s+t_{sv})$ rappresenta la frazione vettorizzabile del codice, che comunemente viene indicata con la lettera **f**, e che la quantità t_{sv}/t_v costituisce il rapporto di velocità

tra l'architettura scalare e vettoriale (*vector/scalar speed ratio*), che qui chiameremo **sr**, la *57* diventa:

$$S_{eff} = \frac{1}{(1 - f) + \frac{f}{sr}} \qquad (58)$$

L'espressione *58* è nota anche come la **legge di Amdahl**. A questo punto è doveroso fare una precisazione a proposito del parametro *sr*, in quanto potrebbe essere confuso con il rapporto del tempo scalare e vettoriale indicato dalla *48*. La differenza tra *sr* e S_n sta nella quantità espressa dalle due coppie di tempi; infatti, mentre s_n e t_n indicano solo il tempo di calcolo dell'operazione vettoriale, t_{sv} e t_v sono dei *tempi effettivi* e quindi comprendono, oltre al tempo di calcolo, anche del tempo di esecuzione delle istruzioni di sezionamento dei vettori, nel caso di macchine provviste di registri vettoriali, e delle istruzioni di controllo del ciclo *DO*.

Naturalmente anche il valore di *sr* varia in funzione dell'operazione aritmetica da eseguire e tende al valore *m* quando la dimensione dei vettori da elaborare è grande.

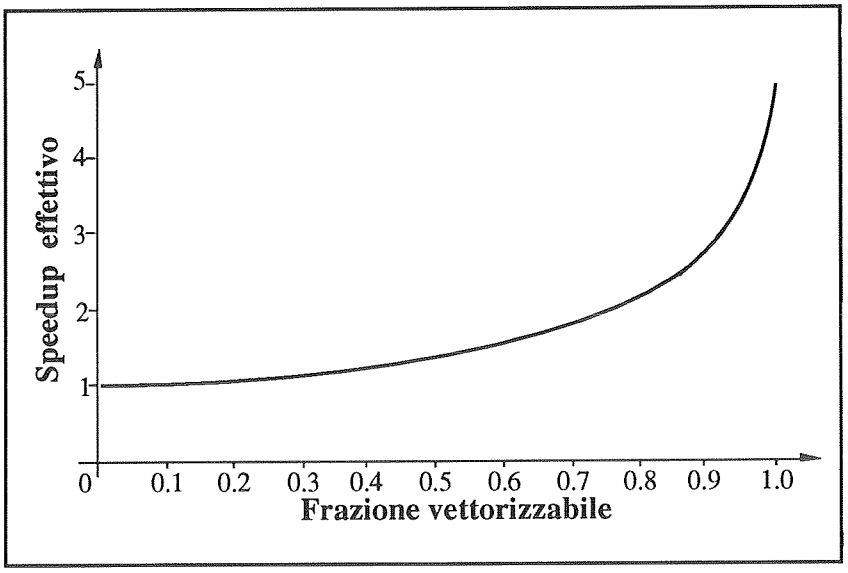


Figura 59 - Andamento dello speedup in funzione della frazione vettorizzabile

La *legge di Amdahl* è importante perchè dimostra che il limite dello *speedup effettivo* è dato sia dalla frazione di codice vettorizzabile, sia dal rapporto di potenza tra architettura scalare e vettoriale, che è un altro elemento caratteristico di una macchina vettoriale. Se supponiamo *sr* uguale a 5 e rappresentiamo l'anda-

mento dello *speedup* in funzione della frazione di codice vettorizzabile si ottiene il grafico mostrato in figura *59*.

La curva indica inequivocabilmente che lo *speedup* inizia ad avere un sensibile incremento quando l'applicazione possiede più del 70% di codice vettorizzabile. Il massimo valore dello *speedup* si ottiene quando f tende 1 e cioè quando:

$$\frac{t_{sv}}{t_s + t_{sv}} \longrightarrow 1 \qquad (60)$$

La condizione *60* si verifica quando t_s è tende a zero, mentre più è grande t_s e più il rapporto della *60* si allontana da 1. Quindi la parte scalare del codice costituisce *una limitazione* delle prestazioni vettoriali di una applicazione.

La previsione dello *speedup*

La legge di Amdahl potrebbe essere utilizzata per stimare lo *speedup* di una applicazione su una determinata macchina vettoriale se fosse nota la frazione vettorizzabile del codice. Poichè questo elemento dipende dalle caratteristiche dell'algoritmo usato, non è possibile sapere a priori quanto tempo di calcolo può essere eseguito in vettoriale, a meno che l'algoritmo non rientri in una categoria nota. Infatti, per quanto riguarda molti campi scientifici sono state analizzate le tipologie degli algoritmi di uso prevalente; la qual cosa ha permesso di classificare le applicazioni a seconda della quantità di codice vettorizzabile e di ottenere quindi i risultati mostrati in figura *61*.

Per ogni disciplina scientifica gli algoritmi più frequentemente usati, che sono indicati nella figura dalla banda nera, non hanno la stessa frazione di codice vettorizzabile ma esiste un intervallo di variabilità la cui maggiore o minore ampiezza rende più o meno difficile la previsione dello *speedup*. La zona tratteggiata rappresenta invece un insieme di algoritmi che hanno un minor impiego nel settore considerato.

Dalle indicazioni della figura *61* è possibile comunque trarre delle conclusioni di ordine generale, come l'affermazione che l'elaborazione delle immagini e la simulazione di fenomeni sismici trovano un buon impiego sulle macchine vettoriali, in quanto i rispettivi codici hanno una frazione vettorizzabile che oscilla tra lo *0.7* e lo *0.9*. Non altrettanto si può dire dei prodotti *CAD* che difficilmente possono ottenere delle buone prestazioni con l'uso della *pipeline*.

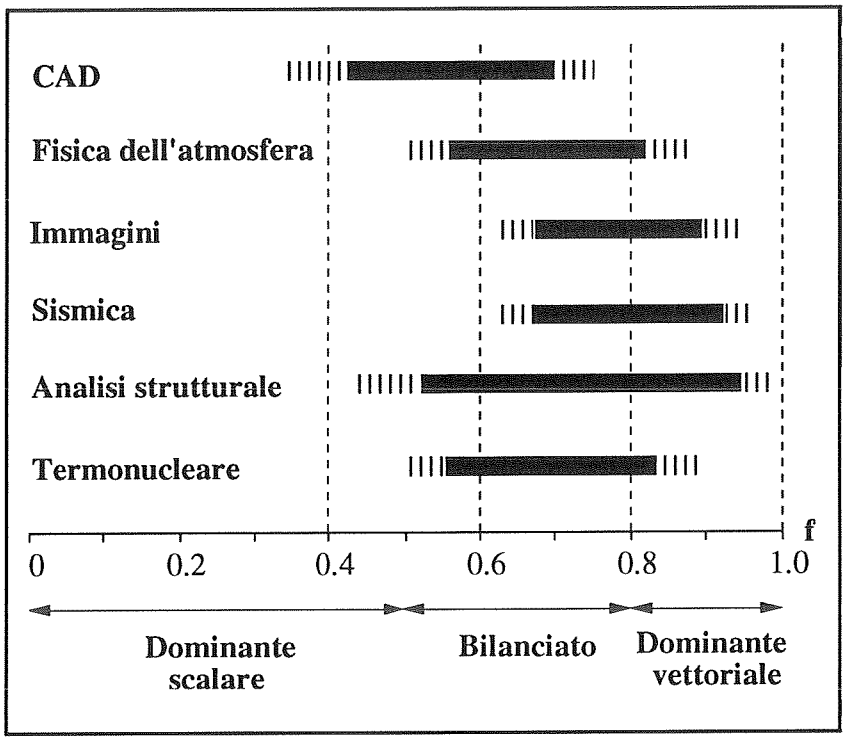


Figura 61 - Frazione vettorizzabile di alcune applicazioni scientifiche

Per le applicazioni che non rientrano in una delle categorie note non si può trarre alcuna indicazione, anche se di massima, circa la frazione di codice vettorizzabile dell'algoritmo usato e quindi dedurre una previsione dello *speedup effettivo*. È comunque possibile, mediante un metodo empirico, ricavare una stima approssimativa dello *speedup*, nel caso in cui l'applicazione sia già funzionante. Questo tentativo, pur introducendo una percentuale di errore non indifferente, non deve essere scartato a priori, poichè la previsione del miglioramento delle prestazioni che si possono ottenere mediante l'uso della *pipeline* rappresenta uno dei principali criteri di scelta delle applicazioni da vettorizzare.

Per illustrare la metodologia usata supponiamo di avere una applicazione la cui esecuzione scalare ha una durata di 100 minuti. Con l'uso di un sottoprogramma che fornisce i tempi di calcolo, si rilevano i tempi spesi nelle zone a più alto tasso di esecuzione, che costituiscono quindi la parte del codice maggiormente candidata alla vettorizzazione. La somma di questi tempi può essere perciò identificata con la frazione del programma t_{sv}, mentre 100-t_{sv} rappresenta la parte scalare dell'applicazione.

Supponendo t_{sv} uguale a 60, il tempo di esecuzione della ipotetica applicazione può essere rappresentato secondo lo

schema di figura *62*.

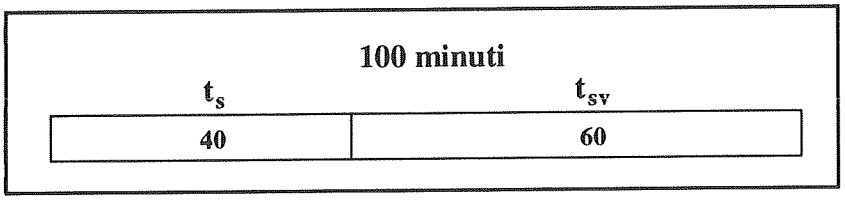


Figura 62 - Esempio di distribuzione del tempo scalare e vettoriale

Nell'ipotesi di lavorare su di una macchina vettoriale che ha un rapporto di velocità tra vettoriale e scalare che oscilla, in base al tipo di operazione eseguita, tra 4 e 6, e che qui per comodità assumiamo essere mediamente uguale a 5, allora il tempo di esecuzione vettoriale è di 52 minuti come si può vedere dalla figura *6*

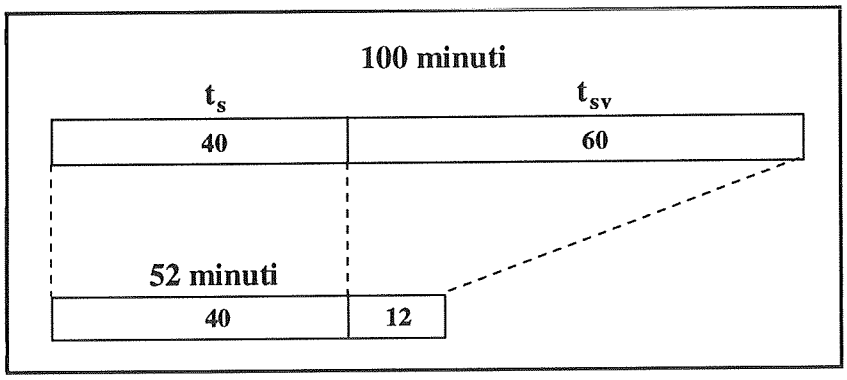


Figura 63 - Previsione del tempo vettoriale su di una macchina con sr=5

Con i dati a disposizione, è possibile ora ricavare lo *speedup* con il seguente rapporto:

$$S_{eff} = \frac{100}{52} = 1.9 \qquad (64)$$

La stima dello speedup della *64* è aderente alla realtà se la parte del codice t_{sv} è vettorizzata al 100%, situazione questa che raramente si verifica. Per ottenere una previsione più esatta è opportuno ricorrere ad una compilazione vettoriale del programma, la quale ci fornisce delle indicazioni sul processo di vettorizzazione. Dall'analisi della diagnostica bisogna tentare di suddividere il co-

dice vettorizzabile nelle seguenti parti:

- La parte vettorizzata
- La parte non vettorizzata, ma che può essere facilmente modificata
- La parte non vettorizzata

Questa suddivisione permette di affinare ulteriormente la previsione del tempo di esecuzione vettoriale. Infatti se supponiamo il codice t_{sv} vettorizzato per una frazione pari al 50% e il restante 50% suddiviso in due parti, di cui il 20% rappresenta il codice che può essere vettorizzato con semplici modifiche mentre il 30% deve essere eseguito in scalare, si ottengono i seguenti tempi:

- Parte vettorizzata: (50% + 20%) di 60 minuti = 42 minuti
- Parte non vettorizzata: 30% di 60 minuti = 18 minuti

Pertanto il tempo vettoriale risultante si ricava nel modo visualizzato nella figura *65*, dove si può notare che, con una percentuale di vettorizzazione più realistica, solo 42 minuti sono soggetti alla riduzione dell'esecuzione mediante l'uso della *pipeline*, mentre l'effettivo tempo scalare passa dai precedenti 40 minuti agli attuali 58 minuti.

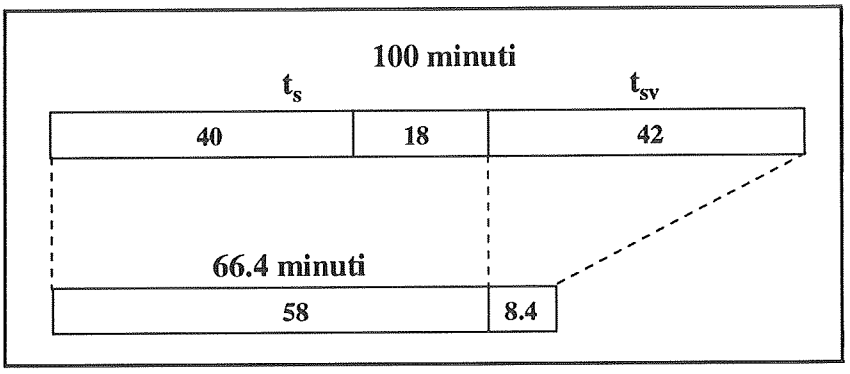


Figura 65 - Previsione "aggiustata" del tempo vettoriale su di una architettura con sr=5

Sotto queste condizioni lo *speedup* diventa:

$$S_{eff} = \frac{100}{66.4} = 1.5 \qquad (66)$$

Il risultato ottenuto con la *66* rappresenta lo *speedup* che ci si deve attendere con l'esecuzione vettoriale dell'applicazione presa in esame. A questo punto l'utente dispone di un buon strumento per decidere se intraprendere o meno la fase di vettorizzazione del programma.

Il valore dello *speedup* ottenuto con il rapporto *66* poteva anche essere ricavato dalla legge di *Amdahl* tenendo presente che la frazione vettorizzabile *f* vale 42/100 e *sr* è uguale a 5. Inserendo infatti questi dati nella *58* si ottiene:

$$S_{eff} = \frac{1}{(1 - 0.42) + \frac{0.42}{5}} = 1.5 \qquad (67)$$

Misura delle prestazioni di una applicazione

Fino ad ora i parametri significativi delle prestazioni di una applicazione, come il tempo scalare o la frazione vettorizzabile, sono stati presunti, ma vediamo adesso come è possibile ottenere dei valori reali. Questa operazione può essere svolta solamente dopo che è terminato il processo di vettorizzazione del programma in quanto uno dei dati richiesti è il tempo di esecuzione vettoriale. Tale elemento costituisce, tra l'altro, anche la causa principale dell'impossibilità di effettuare una previsione accurata dello *speedup* con il solo tempo di esecuzione scalare.

Come appena detto, la fase della valutazione delle prestazioni inizia dopo la fine dell'esecuzione scalare e vettoriale dell'applicazione. Generalmente tutti i sistemi forniscono delle indicazioni più o meno dettagliate sui tempi spesi nelle varie fasi dell'esecuzione di un programma, ma, per le nostre finalità, gli unici dati richiesti sono il tempo di lavoro del processore scalare e quello del processore vettoriale. Dopo le due esecuzioni avremo pertanto una situazione come quella mostrata in figura *68*.

Ovviamente, poichè nell'esecuzione scalare viene usato un solo processore, il tempo fornito dal sistema coincide con il tempo dell'intera l'esecuzione, mentre nella versione vettoriale il tempo totale T_v è dato dalla somma dei tempi di lavoro di entrambi i processori. La conoscenza di questi due elementi consente di ottenere un indice di prestazione importante che è lo *speedup effettivo,* il quale già fornisce un'idea chiara del grado di

adattabilità dell'algoritmo all'architettura vettoriale.

	Tempo Processore Scalare	Tempo Processore Vettoriale
Esecuzione Scalare	T_s	0
Esecuzione Vettoriale	t_s	t_v

Figura 68 - Dati dei tempi dell'esecuzione scalare e vettoriale

Per avere, però, un quadro più completo delle caratteristiche dell'applicazione è necessario scendere maggiormente nel dettaglio, individuando la frazione vettorizzabile del programma. Tenendo presente lo schema di figura *55* è possibile risalire al tempo t_{sv} mediante una semplice sottrazione, poichè i termini T_s e t_s sono noti:

$$t_{sv} = T_s - t_s \qquad (69)$$

A questo punto la frazione vettorizzabile f è ottenuta come rapporto tra t_{sv} e T_s. Qualcuno potrebbe obiettare che la frazione vettorizzabile del programma dipende dalla capacità del programmatore nel trasformare il codice scalare in un codice vettoriale e non è una caratteristica intrinseca dell'applicazione. Questa osservazione è, in parte, vera e cioè le caratteristiche dell'*oggetto* che andiamo a valutare sono, in un certo senso, condizionate dal nostro operato. Però è stato assunto che il tempo T_v si riferisca al miglior algoritmo vettoriale e questo presuppone che il lavoro di vettorizzazione si stato svolto al meglio delle nostre possibilità.

Comunque se la frazione di codice vettorizzabile non è soddisfacente può darsi che l'algoritmo non sia idoneo ad una architettura vettoriale oppure sia necessario intervenire sull'applicazione con modifiche più complesse.

Bibliografia

G. Amdahl, G.A. Blaauw, F.P. Brooks, "Architecture of the IBM System/360", *IBM Journal of Research and Development*, n. 8, 1964, pp. 87-101

G. Amdahl, "The validity of the single processor approach to achieving large scale computing capabilities", *AFIPS Conference Proceedings*, vol. 30, 1967

C. Arnold, "Vector optimization on the CYBER 205", *International Conference on Parallel Processing Proceedings*, Silver Spring, 1983, pp. 530-536

F. Baiardi, *Linguaggi per la programmazione concorrente*, Franco Angeli, Milano, 1985

F. Baiardi, A. Tomasi, M. Vanneschi, *Architettura dei sistemi di elaborazione*, Franco Angeli, Milano, 1987

R. Baraglia, D. Laforenza, R. Perego, "Caratterizzazione delle prestazioni vettoriali dell'elaboratore IBM 3090-VF", *Rivista di informatica AICA*, vol. XXI, n. 1, 1991, pp. 59-72

S. Barsocchi, R. Ferrini, P. Lazzareschi, R. Medves, P. Pancrazi, *Introduzione al calcolo vettoriale sull'IBM 3090-VF*, Rapporto interno CNR C87-29, Pisa, 1987

R. Bianchi Bandinelli, R. Ferrini, D. Laforenza, P. Lazzareschi, R. Moia, *Relazione e documentazione Benchmark IBM 3090/200 VF CNR - CNUCE (Pisa)*, Rapporto interno CNR C87-37, Pisa, 1987

D.A. Calahan, W.G. Ames, "Vector processors: models and applications", *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, 1979, pp. 715-726

R.S. Clark, T.L. Wilson, "Vector System performance of the IBM 3090", *IBM systems Journal*, vol. 25, n. 1, 1986, pp. 63-82

CNUSC, *Stage de formation au calcul vectoriel sur IBM 3090 VF*, Centre National Universitaire Sud de Calcul, Montpellier, 1986

J.J. Dongarra, "Performance of various computers using standard linear equations software in a Fortran environment", *Argonne National Laboratory Techinal Memorandum n. 23*, 1989

R. Ferrini, *Il calcolo vettoriale*, Rapporto interno CNR C88-24, Pisa, 1988

R.W. Hockney, C.R. Jesshope, *Parallel Computers*, Adam Hilger Ltd, Bristol, 1983

R.W. Hockney, C.R. Jesshope, *Parallel Computers 2*, Adam Hilger Ltd, Bristol, 1988

R.W. Hockney, "Performance Parameters and Benchmarking of Supercomputers", *Parallel Computing*, Vol. 17, 1991, pp. 1111-1190

K. Hwang, F.A. Briggs, *Computer Architecture and Parallel Processing*, Mc Graw Hill, New York, 1984

K. Hwang, *Supercomputers: Design and Applications*, IEEE Computer Society Press, Silver Spring, 1984

R.N. Ibbett, *The Architecture of High Performance Computers*, Macmillan, Londra, 1982

IBM, *IBM 3090 Engineering/Scientific Performance*, IBM Corporation., Form Number GG66-0245

IBM, *IBM System/370 Vector Operation*, IBM Corporation., Form Number SA22-7125-2, 1987

P.M. Johnson, "An Introduction to Vector Processing", *Computer Design*, vol. 17, n. 2, 1978, pp. 89-97

M.J. Kascic, "Vector processing on the CYBER 200", *Infotech State of the Art Report: Supercomputers*, vol. 2, 1979, pp. 237-270

K. Kennedy, "A survey of code optimization techniques", *Le point sur la Compilation*, 1978, pp. 115-163

K. Kennedy, *Automatic translation of Fortran programs to vector form*, Rice Technical Report 476-029-4, Houston, 1980

E.W. Kozdrowicki, D.J. Theis, "Second Generation of Vector Supercomputers", *Computer*, 1980, pp. 71-83

P.M. Kogge, *The Architecture of pipelined Computers*, Mc Graw Hill, New York, 1981

D.J. Kuck, *The Structure of Computers and Computations*, John Wiley and Sons, New York, 1978

D. Laforenza, "Classificazione e trend architetturali", *Corso di Formazione: Architetture avanzate per il Calcolo Scientifico*, Pisa, 1985

D. Laforenza, "Elaborazione Parallela e Calcolo Vettoriale", *Informatica Oggi*, anno 6, n. 13, 1986, pp. 42-71

R. Moia, "Strumenti software dell'ambiente vettoriale", *Corso di Formazione: Tecniche di programmazione degli elaboratori vettoriali della serie IBM 3090*, Pisa, 1987

Pacific-Sierra, *Efficient Fortran techniques for vector processors*, Pacific Sierra Research Corporation, 1983

D.A. Padua, M. Wolfe, "Advanced compiler optimizations for supercomputer", *Communications of ACM*, 1986, pp. 1184-1201

K. Radecki, *Introduction to Processor Performance Evaluation*, IBM Washington Systems Center Technical Bulletin n. GG66-0232, IBM Corporation, 1986

C.V. Ramamoorthy, H.F. Li, *Pipeline architecture*, Computer Survey, n. 9, 1977, pp. 61-102

R.M. Russell, "The CRAY-1 Computer System", *Communications of the ACM*, vol. 21, n. 1, 1978, pp. 63-72

W. Schönauer, *Scientific Computing on Vector Computers*, North-Holland, Amsterdam, 1987

P. Sguazzero, *Vector and Parallel Processors for scientific Computation*, Accademia Nazionale dei Lincei, Roma, 1986

Y. Singh, G.M. King, J.W. Anderson, "IBM 3090 performance: A balanced system approach", *IBM systems Journal*, vol. 25, n. 1, 1986, pp. 20-35

D.J. Theis, "Vector Supercomputers", *Computers*, vol. 7, n. 4, 1974, pp. 52-61

S.A. Williams, "The portability of programs and languages for vector and array processors", *Infotech State of the Art Report: Supercomputers*, vol. 2, 1979, pp. 382-393