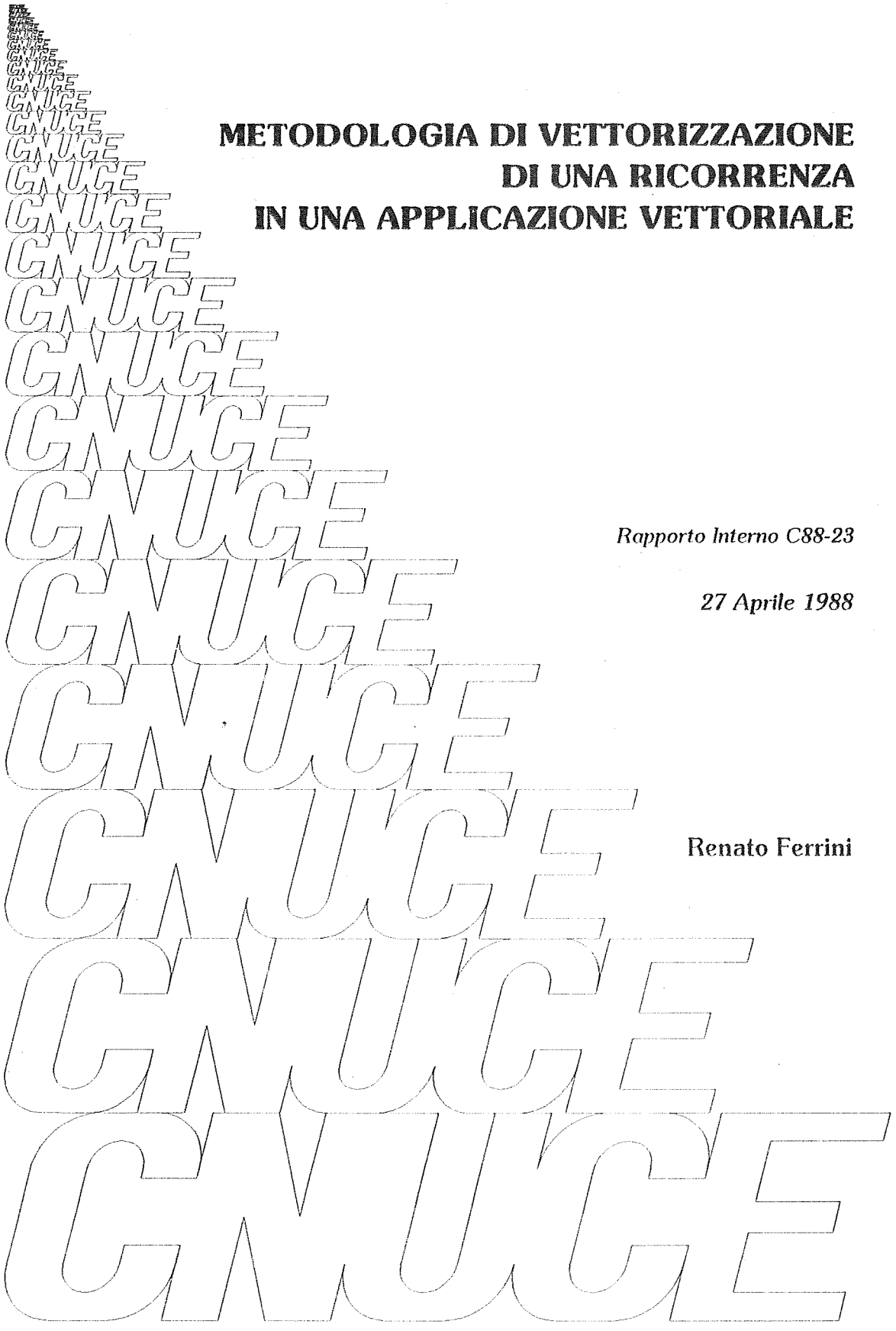

METODOLOGIA DI VETTORIZZAZIONE DI UNA RICORRENZA IN UNA APPLICAZIONE VETTORIALE

Rapporto Interno C88-23

27 Aprile 1988

Renato Ferrini

METODOLOGIA DI VETTORIZZAZIONE DI UNA RICORRENZA IN UNA APPLICAZIONE VETTORIALE

27 Aprile 1988

Renato Ferrini

CNUCE - Istituto del CNR
via Santa Maria 36
56100 PISA (Italy)

Prefazione

Nell'ambito del Progetto Finalizzato *Sistemi Informatici e Calcolo Parallelo* sono stati condotti degli studi in vari settori del Calcolo Vettoriale. Una particolare indagine e' stata rivolta alla possibilita' di eliminazione di alcuni inibitori presenti in un programma Fortran.

In questo documento sono riportati gli studi effettuati sull'inibitore di *ricorrenza* che costituisce un elemento tra i piu' comuni presente in una applicazione scientifica.

Contenuto

Figure

La ricorrenza

Definizione della ricorrenza

Una delle cause piu' frequenti di inibizione della vettorizzazione che si incontrano in una applicazione Fortran e' rappresentata dalla *ricorrenza*. Questo inibitore e' un tipo particolare di Data Dependence che si manifesta per il riutilizzo di uno stesso elemento di un vettore o di una matrice in iterazioni diverse di un ciclo DO.

La ricorrenza puo' avvenire per la presenza, nel codice Fortran, di particolari elementi di programmazione, quali, ad esempio, l'uso di un indice ausiliario, l'uso di un indirizzamento non lineare oppure l'uso di un indirizzamento indiretto, ecc...

Ma la forma piu' frequente in cui compare una ricorrenza e' del tipo:

```
      DO  1  I = 1, N
      A(I+INC) = A(I) + B(I)
1     CONTINUE
```

Se l'incremento *INC* e' un numero intero minore od uguale a zero, allora la ricorrenza determina una dipendenza di tipo *Forward* ed in questo caso il ciclo puo' essere eseguito in modo vettoriale. Anche se il compilatore, non essendo noto il valore di *INC*, opta per l'esecuzione scalare, il programmatore puo' forzare la vettorizzazione del ciclo mediante l'uso di una direttiva.

Se, invece, la variabile *INC* contiene un valore positivo, la ricorrenza e' di tipo *Backward*. Infatti, in questo caso, l'indice del vettore A che si trova alla sinistra del segno = e' maggiore dell'indice dello stesso vettore posto alla destra del segno = e cio' determina, nel corso delle iterazioni del ciclo, la definizione di elementi che vengono successivamente riutilizzati nel calcolo dell'espressione.

Tipologia delle ricorrenze

A volte, in un ciclo DO, la ricorrenza si manifesta per un'alterazione dell'indice del vettore che si trova alla destra del segno = come e' mostrato nel seguente esempio:

```
      DO  1  I = 2, N
      A(I) = A(I-1) + B(I)
1     CONTINUE
```

In questi casi, pero', il codice puo' essere sempre ricondotto nella forma presentata nel primo esempio. Infatti il precedente codice puo' essere cosi' riscritto:

```
      DO  1  I = 1, N-1
      A(I+1) = A(I) + B(I+1)
1     CONTINUE
```

Pertanto d'ora in avanti faremo sempre riferimento ad un costrutto del tipo:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I)
1     CONTINUE
```

il quale presenta un'alterazione dell'indice del vettore che si trova alla sinistra del segno uguale.

Vettorizzazione di una ricorrenza

Frazionamento del ciclo

Ovviamente il riutilizzo degli stessi elementi in iterazioni successive impedisce l'esecuzione parallela del calcolo degli elementi e per tale motivo il ciclo DO deve essere eseguito in modo scalare. Lo scopo di questo documento e' quello di studiare le possibilita' di eliminare tale inibitore.

Innanzitutto e' ormai noto che esiste una tecnica, chiamata *Frazionamento del ciclo* che permette di isolare l'istruzione che presenta una ricorrenza. Ad esempio il seguente ciclo:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I)
      C(I) = A(I) + B(I)
1     CONTINUE
```

puo' essere frazionato in due cicli, dei quali il secondo viene vettorizzato:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I)
1     CONTINUE
      DO  2  I = 1, N
      C(I) = A(I) + B(I)
2     CONTINUE
```

Limiti del frazionamento

Questa tecnica e' pero' applicabile solo nel caso in cui esistano due o piu' istruzioni indipendenti all'interno del ciclo. Inoltre l'istruzione che contiene la ricorrenza continua ad essere eseguita in modo scalare e cio' rappresenta un grosso svantaggio, soprattutto se nell'istruzione e' presente un'espressione complessa, il cui calcolo richiede molti cicli di macchina.

E' interessante percio' cercare di individuare un metodo che permetta di trasformare il codice originale in un altro semanticamente equivalente e che sia interamente vettorizzabile.

Trasformazione del codice

Per il raggiungimento del suddetto obiettivo e' necessario analizzare lo sviluppo dell'esecuzione. Come esempio viene utilizzato il codice gia' visto:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I)
1     CONTINUE
```

la cui esecuzione prevede:

```
A(2) = A(1) + B(1)
A(3) = A(2) + B(2) = A(1) + B(1) + B(2)
A(4) = A(3) + B(3) = A(1) + B(1) + B(2) + B(3)
     .
     .
     .
```

Lo sviluppo dell'esecuzione scalare mette in evidenza una relazione interessante per la quale l'*i-esimo* elemento del vettore A e' dato dalla somma del primo elemento con la sommatoria dei primi *I* elementi del vettore B.

Quanto detto puo' essere sintetizzato con la seguente formula, tenendo conto che al posto di B(I) puo' trovarsi una qualsiasi espressione:

$$a_{I+1} = a_1 + \sum_{K=1}^{I} < espressione >_K$$

Espressione costante

Eliminazione della ricorrenza

Se l'espressione e' costante, cioe' contiene un valore assoluto oppure un elemento di un vettore o di una matrice che non dipende dall'indice del ciclo DO, si ottiene la seguente relazione:

$$a_{I+1} = a_1 + \sum_{K=1}^{I} c$$

che puo' essere riscritta come:

$$a_{I+1} = a_1 + I \times c$$

In base a quanto detto e' possibile riscrive il codice:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + C
1     CONTINUE
```

sotto la forma:

```
      DO  1  I = 1, N
      A(I+1) = A(1) + I * C
1     CONTINUE
```

Poiche' il nuovo codice non contiene piu' la ricorrenza, diventa vettorizzabile e pertanto offre delle migliori prestazioni.

Valutazione delle prestazioni

Per valutare il guadagno ottenuto e' stato utilizzato lo *Speedup*, che e' dato dal rapporto tra il tempo di esecuzione del ciclo scalare ed il tempo di esecuzione del ciclo vettoriale. Indicando T_s il tempo

scalare e T_v il tempo vettoriale, lo Speedup puo' essere rappresentato come:

$$Speedup = \frac{T_s}{T_v}$$

Le misure dei due tempi sono state eseguite su di un IBM 3090-180E provvisto della Feature Vettoriale. Poiche' le prestazioni vettoriali dipendono dalla dimensione del vettore, e quindi dal numero degli elementi da elaborare, e' stato ritenuto utile calcolare lo Speedup al variare di tale fattore. Inoltre per ragioni di praticita' e' stato fissato il limite massimo del vettore a 512.

In Figura 1 e' riportato lo Speedup ottenuto con l'esecuzione del ciclo in scalare e dell'equivalente codice vettorizzato.

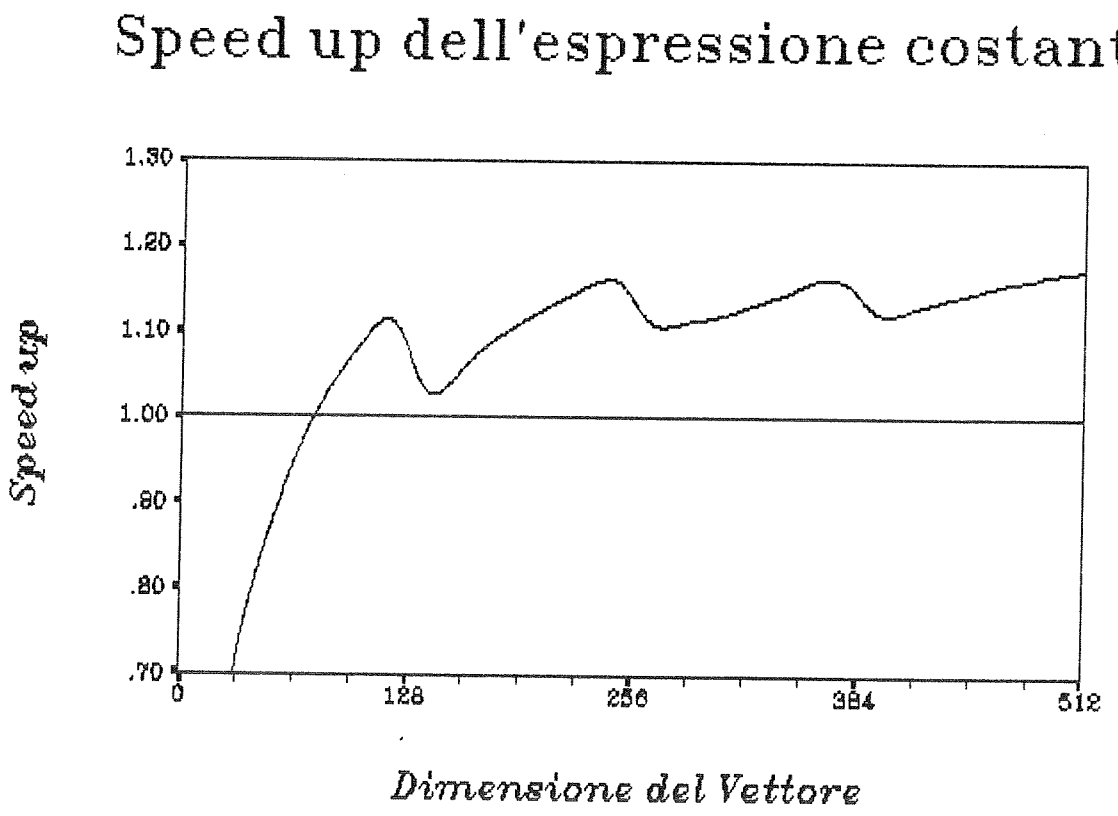

Figura 1. Speedup con l'espressione costante

Il grafico ottenuto presenta un andamento non lineare della curva dello Speedup e questo fenomeno e' dovuto al *sezionamento* del vettore durante l'esecuzione vettoriale. Infatti nell'IBM 3090-VF l'elaborazione degli elementi avviene a gruppi di 128 *(Section Size)*

in quanto questo valore rappresenta la dimensione massima di un registro vettoriale.

Quindi se da una parte l'aumento del numero di elementi del vettore da calcolare fa crescere le prestazioni vettoriali rispetto a quelle scalari, dall'altra il superamento della sezione produce una brusca degradazione dello Speedup. Infatti in questo caso la seconda sezione del vettore e' troppo piccola e cio' aumenta notevolmente il tempo di esecuzione vettoriale, mentre il tempo scalare continua ad avere una crescita lineare.

Il verificarsi di questo fenomeno e' presente, anche se man mano in misura minore, per tutte le dimensioni del vettore che siano multiple di 128, come e' ampiamente visibile dal grafico.

Per quanto riguarda l'analisi dei risultati, si puo' notare che, con valori piccoli della dimensione del vettore, lo Speedup si mantiene al di sotto di 1 e cio' significa che il tempo scalare e' minore di quello vettoriale. Questo risultato non deve meravigliare in quanto il tempo di *startup* della *pipeline* ha incidenza molto alta se il numero di elementi da elaborare e' basso.

Comunque il grafico visualizza uno Speedup di circa 1.2 quando la dimensione del vettore e' 512. Quindi in questo caso con l'esecuzione vettoriale si ottiene un guadagno del 20% rispetto all'esecuzione scalare.

Espressione variabile

Eliminazione della ricorrenza

Se invece l'espressione non e' costante, ma contiene alcuni elementi che dipendono dall'indice del ciclo, allora non e' possibile operare la trasformazione del ciclo precedentemente vista. Pertanto e' necessario riscrivere il ciclo seguendo la formula estesa. Il seguente codice:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I)
1     CONTINUE
```

diventa percio':

```
      DO  1  I = 1, N
      A(I) = A(1)
      DO  2  K = 1, I
      A(I) = A(I) + B(K)
2     CONTINUE
1     CONTINUE
```

Come si puo' notare il ciclo piu' interno del codice trasformato diventa vettorizzabile.

Valutazione delle prestazioni

Pero' tale trasformazione non risulta conveniente, poiche' l'analisi dell'esecuzione dei due cicli dimostra che il tempo scalare e' sempre minore del tempo vettoriale. La valutazione e' stata effettuata considerando il tempo di esecuzione dei cicli DO come la somma del tempo per l'accesso ai dati con i cicli macchina necessari per il calcolo.

Prestazione del ciclo scalare

In base a cio', per il ciclo scalare, si ottiene:

- *Accesso ai dati*

  Per il calcolo dell'espressione contenuta nel ciclo, sono necessari i seguenti accessi alla memoria:

  1. 1 accesso per il caricamento dell'elemento A(1). Nelle iterazioni successive non si verificano altri accessi alla memoria poiche' i vari elementi A(2), A(3), ecc... sono stati calcolati all'iterazione precedente e quindi risiedono in un registro scalare.
  2. N accessi alla memoria per il caricamento degli elementi del vettore B.
  3. N accessi alla memoria per la memorizzazione degli elementi A(I+1).

  Il totale degli accessi risulta pertanto essere:

  $$1 + N + N = 2 \times N + 1 \ accessi$$

- *Tempo per il calcolo dell'operazione*

  Come gia' detto, il tempo di calcolo viene espresso in termini di cicli di macchina. Poiche' l'operazione di somma scalare richiede 4 cicli macchina, la somma di N elementi durera':

  $$4 \times N \ cicli\ macchina$$

Prestazione del ciclo vettoriale

Per quanto riguarda il ciclo vettoriale si ottengono, invece, i seguenti valori:

- *Accesso ai dati*

  Per quanto riguarda il tempo per l'accesso ai dati, si e' ritenuto opportuno dividere l'operazione in due fasi:

  - calcolo degli accessi del ciclo piu' interno
  - calcolo degli accessi totali in base alla ripetizione del ciclo piu' interno

  Nel ciclo piu' interno si hanno i seguenti accessi:

  1. 1 accesso per il caricamento dell'elemento A(I). Poiche' per l'esecuzione vettoriale, il valore A(I) viene espanso in tutto il registro vettoriale, non sono necessari altri accessi.
  2. *I* accessi alla memoria per il caricamento degli elementi del vettore B.

3. 1 accesso alla memoria per la memorizzazione dell'elemento A(I).

Il numero totale di accessi alla memoria del ciclo piu' interno risulta pertanto:

$$1 + I + 1 = I + 2 \ \textit{accessi}$$

Poiche' il ciclo piu' interno viene ripetuto da 1 a N volte, il numero totale di accessi e' dato dalla seguente formula:

$$\sum_{I=1}^{N} (I + 2)$$

che puo' essere riscritta come:

$$\sum_{I=1}^{N} I + \sum_{I=1}^{N} 2$$

Considerando N pari, la prima sommatoria e' uguale a:

$$\sum_{I=1}^{N} I = (N + 1) \times \frac{N}{2}$$

Mentre la seconda sommatoria risulta essere:

$$\sum_{I=1}^{N} 2 = 2 \times N$$

In totale avremo percio':

$$2 \times N + (N + 1) \times \frac{N}{2} \ \textit{accessi}$$

- *Tempo per il calcolo dell'operazione*

Come per l'accesso ai dati, anche per il calcolo del tempo di esecuzione si procede prima per il ciclo piu' interno e poi si estende all'intero codice. Poiche' la somma vettoriale ha una durata di $3 + K$ cicli macchina, dove K e' il numero di elementi del vettore, in questo caso avremo $3 + I$ cicli macchina. Ripetendo tale operazione da 1 a N volte si ottiene:

$$\sum_{I=1}^{N} (I + 3)$$

Considerando N pari ed operando le trasformazioni viste in precedenza, si ottiene un risultato finale di:

$$3 \times N + (N + 1) \times \frac{N}{2} \quad \textit{cicli macchina}$$

Confronto tra le prestazioni scalari e vettoriali

Si procede adesso al raffronto degli accessi alla memoria ed al tempo di calcolo nei due casi:

- *Confronto degli accessi alla memoria*

  Per raggiungere tale finalita' si calcola il rapporto tra il numero degli accessi del ciclo scalare con quelli del ciclo vettoriale, ottenendo:

$$\frac{2 \times N + 1}{2 \times N + (N + 1) \times \frac{N}{2}}$$

  che operando le opportune trasformazioni si riduce a:

$$\frac{4 \times N + 2}{4 \times N + (N + 1) \times N}$$

  Si puo' notare che la differenza e' data dalla parte eccedente i $4 \times N$ accessi alla memoria che sono presenti in entrambi i casi. Ma mentre nel ciclo scalare questa quantita' e' costante, e pari a 2, nel ciclo vettoriale e' un valore crescente in funzione della dimensione del vettore.

  In ogni caso i tempi di accesso sono piu' favorevoli nel ciclo scalare in quanto solo quando N e' uguale a 1 il tempo scalare coincide con il tempo vettoriale, mentre, negli altri casi, il primo e' sempre inferiore al secondo.

- *Confronto dei tempi di calcolo*

  Anche in questo caso si procede al rapporto tra il tempo di calcolo scalare e quello vettoriale, che risulta essere:

$$\frac{4 \times N}{3 \times N + (N + 1) \times \frac{N}{2}}$$

che, con le opportune riduzioni, diventa:

$$\frac{8}{6 + (N + 1)}$$

Anche in questo caso il tempo scalare e' inferiore al tempo vettoriale come e' evidente dalla formula per valori di N maggiori di 1.

Conclusioni sulle prestazioni

Come gia' affermato, il ciclo vettoriale risulta piu' lento di quello scalare e cio' e' dimostrato dalle valutazioni effettuate, che portano percio' ad escludere l'utilizzo di tale trasformazione del codice in quanto risulta piu' sfavorevole. Cio' appare ancora piu' evidente se si tiene anche conto del tempo delle istruzioni scalari che, nel caso del ciclo vettoriale, sono necessarie per l'inizializzazione della pipeline e per il controllo del ciclo DO piu' interno.

Lo svantaggio, in termini di tempo di esecuzione, apportato dalla riscrittura del ciclo, consiste nel fatto che per il calcolo di un elemento A(I+1), mentre nel ciclo scalare ad ogni iterazioni viene fatto uso del risultato dell'iterazione precedente, nel ciclo vettoriale ogni volta e' necessario calcolare tutta la sommatoria degli elementi B(I). Cio' comporta un aumento notevole di operazioni aritmetiche e di conseguenza un aumento del tempo di calcolo, che non e' compensato dal risparmio dell'esecuzione vettoriale.

Trasformazione parziale

Frazionamento dell'istruzione

Dati i risultati negativi ottenuti con la riscrittura di un ciclo contenente una ricorrenza in un codice vettorizzabile, e' stata valutata la possibilita' di risolvere parzialmente il problema. Una soluzione del problema puo' consistere nel continuare ad usare il risultato ottenuto all'iterazione precedente per il calcolo degli elementi A(I+1) e di rivolgere l'attenzione solo all'elaborazione dell'espressione.

Infatti, come gia' detto, lo svantaggio dell'esecuzione scalare di un ciclo DO rispetto a quella vettoriale e' tanto maggiore quanto piu' e' complessa l'espressione contenuta nell'istruzione. Pertanto si puo' pensare di modificare il codice solo in quei casi dove il ricorso all'uso della *pipeline* offre delle buone prestazioni.

In tale ottica si puo' usare l'espediente di vettorizzare solo il calcolo dell'espressione, lasciando inalterata la ricursione per la determinazione degli elementi del vettore. In pratica si viene a suddividere l'istruzione in due parti e per tale motivo tale tecnica puo' essere chiamata **Frazionamento dell'istruzione.**

Il funzionamento del meccanismo risulta maggiormente chiaro se si considera il seguente esempio:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I) * C(I)
1     CONTINUE
```

Gli elementi A(I+1) sono calcolati partendo dagli elementi A(I) e aggiungendo a questi ultimi il risultato dell'espressione $B(I) \times C(I)$. La tecnica del frazionamento dell'istruzione consiste, quindi, nel calcolarsi preventivamente la suddetta espressione e di sommare solamente il risultato agli elementi A(I).

La trasformazione prevede percio' la creazione di un nuovo ciclo DO, completamente vettorizzabile, in cui si provvede al calcolo dell'espressione. I risultati parziali possono essere memorizzati negli elementi A(I+1), evitando cosi' la creazione di un vettore temporaneo che comporterebbe l'utilizzo di una maggiore quantita'

di memoria e che potrebbe introdurre dei tempi maggiori di accesso ai dati.

Il precedente codice puo' quindi essere cosi' riscritto:

```
      DO  1  I = 1, N
      A(I+1) = B(I) * C(I)
1     CONTINUE
      DO  2  I = 1, N
      A(I+1) = A(I) + A(I+1)
2     CONTINUE
```

Valutazione delle prestazioni

E' difficile stabilire quale sia il guadagno del tempo di esecuzione del primo codice rispetto al nuovo codice, poiche' cio' dipende dalle caratteristiche dell'espressione. Sicuramente se nell'espressione non esistono delle operazioni aritmetiche, la trasformazione non apporta nessun miglioramento.

Espressione senza operazioni aritmetiche

Se infatti si analizza il seguente ciclo:

```
      DO  1  I = 1, N
      A(I+1) = A(I) + B(I)
1     CONTINUE
```

si puo' notare che l'espressione e' ridotta ad una operazione di assegnamento. Pertanto la riscrittura del codice sotto la forma:

```
      DO  1  I = 1, N
      A(I+1) = B(I)
1     CONTINUE
      DO  2  I = 1, N
      A(I+1) = A(I) + A(I+1)
2     CONTINUE
```

ovviamente aumenta il tempo di esecuzione, in quanto viene effettuata un'inutile e costosa assegnazione degli elementi B(I) in A(I+1), svolta dal primo ciclo. In questo caso il frazionamento dell'istruzione non risulta conveniente e pertanto e' vantaggioso eseguire il ciclo in modo scalare.

Espressione con una sola operazione aritmetica

Poiche' risulterebbe impossibile calcolare le prestazioni dei cicli DO contenenti tutte le diverse espressioni, si e' ritenuto opportuno valutare lo Speedup delle 5 operazioni aritmetiche fondamentali. Tale scelta e' stata motivata dal fatto che, partendo da tali risultati, qualunque sia la complessita' dell'espressione, e' possibile stimare approssimativamente il guadagno delle prestazioni ottenibili con il frazionamento dell'istruzione.

I tempi sono stati ricavati seguendo le stesse modalita' gia' descritte per il calcolo dello Speedup di un'espressione costante. Un elemento da tener presente e' che l'istruzione vettoriale che viene usata per eseguire l'operazione aritmetica sugli elementi dei vettori A e B e' del tipo *Register to Memory*.

In Figura 2 sono visualizzati gli Speedup dei cicli DO contenenti un'espressione in cui e' presente una sola operazione aritmetica.

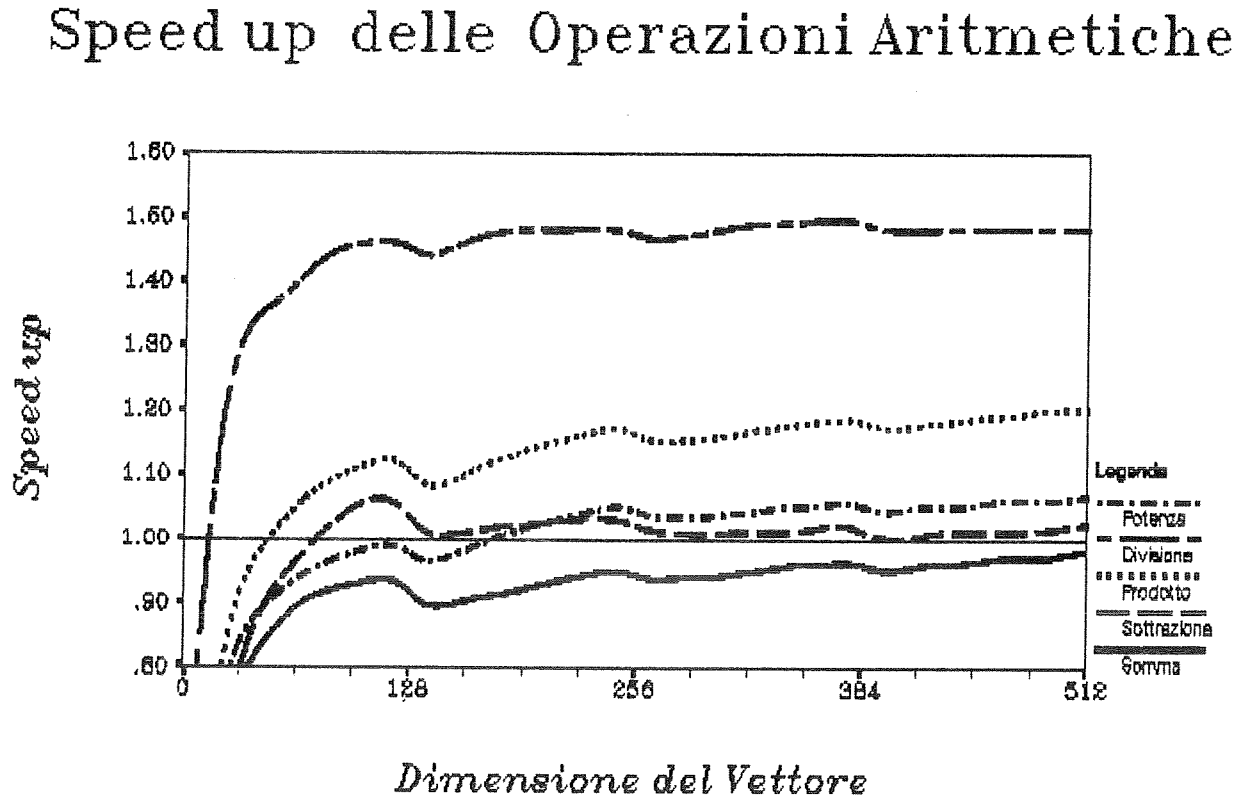

Figura 2. Speedup delle operazioni aritmetiche

Dalla figura risulta subito evidente che se nell'espressione e' presente una sola somma e' piu' conveniente il codice originale. Cio' e' giustificabile dal fatto che tale operazione richiede pochi cicli macchina e questo non compensa il tempo speso dalle istruzioni

scalari per la gestione del ciclo DO aggiunto e dal maggior numero degli accessi ai dati. Se pero' la somma e' combinata con un'altra operazione aritmetica allora le prestazioni del codice vettoriale risultano migliori di quello scalare.

I vantaggi piu' elevati si ottengono con le operazioni di prodotto e divisione, in quanto l'espletamento di tali operazioni richiede l'uso di parecchi cicli macchina. In questi casi lo Speedup supera il valore di 1 gia' con una dimensione del vettore di poco al di sotto di 32 elementi e nel caso della divisione si arriva ad un valore di 1.5 con N uguale a 512.

Le restanti operazioni aritmetiche, e cioe' la sottrazione e l'elevamento a potenza, offrono dei vantaggi solo se il numero degli elementi da elaborare e' superiore a 128.

Espressione con piu' operazioni aritmetiche

Naturalmente se l'espressione contiene piu' di un'operazione aritmetica il vantaggio dell'esecuzione vettoriale e' ancora maggiore rispetto all'esecuzione scalare.

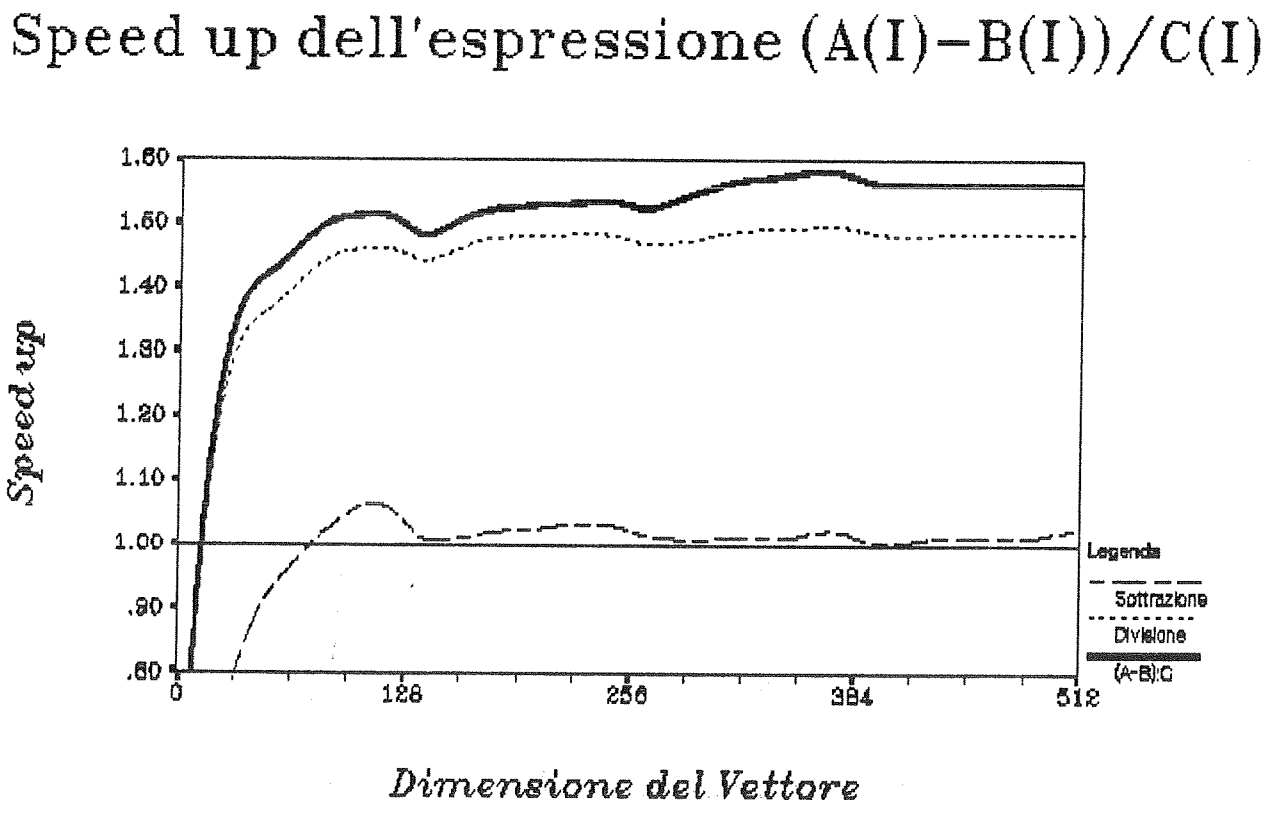

Figura 3. Speedup dell'espressione (A-B)/C

Per verificare quanto detto e' stata presa come esempio la seguente espressione:

$$(A(I) - B(I)) / C(I)$$

di cui e' stato calcolato lo Speedup.

In Figura 3 sono mostrati i risultati ottenuti con l'esecuzione dell'espressione in scalare ed in vettoriale.

Sul grafico sono stati riportati gli Speedup delle singole operazioni della sottrazione e della divisione, per evidenziare meglio il guadagno ottenuto dall'espressione che le contiene entrambe. Da un'analisi piu' approfondita si nota che non esiste una relazione precisa tra lo Speedup delle operazioni aritmetiche usate singolarmente e combinate tra di loro. La motivazione di tale fenomeno e' dovuta a fattori quali il riuso dei dati nei registri vettoriali o un diverso accesso agli elementi in memoria.

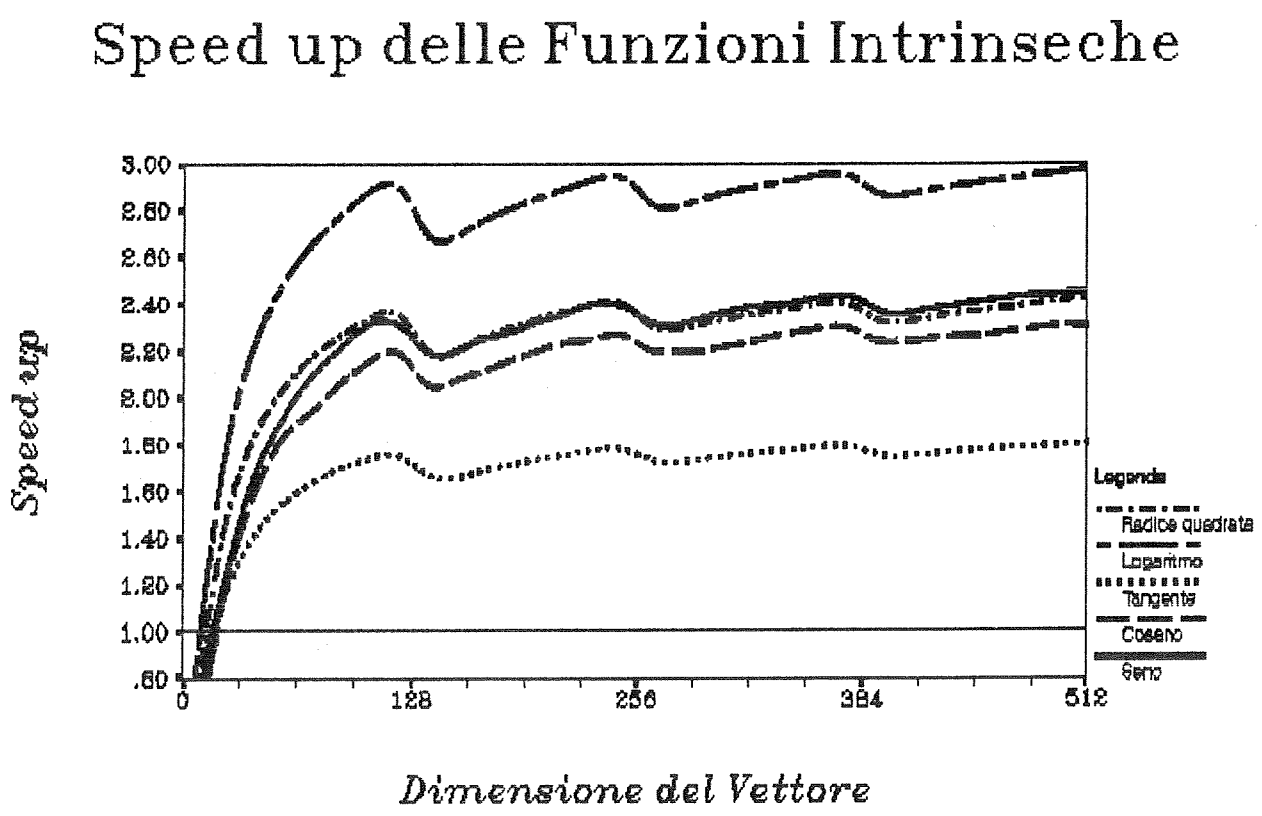

Figura 4. Speedup delle funzioni intrinseche

Espressione con funzioni intriseche

I maggiori vantaggi si ottengono con la presenza di funzioni intrinseche all'interno dell'espressione.

Com'e' noto esiste una versione vettoriale delle principali funzioni intrinsiche ed il loro uso in un ciclo vettorizzabile non costituisce un inibitore.

Poiche' il calcolo di una di tali funzioni richiede molti cicli macchina, la versione vettoriale offre delle prestazioni notevolmente migliori rispetto alla versione scalare. Pertanto in questi casi la tecnica del frazionamento dell'istruzione e' sicuramente vantaggiosa, anche se l'espressione e' costituita da una sola funzione intrinseca.

In Figura 4 sono stati riportati gli Speedup delle principali funzioni intrinseche.

Il grafico mostra chiaramente che lo Speedup e' di un altro ordine di grandezza rispetto a quello delle operazioni aritmetiche. Infatti con la funzione intriseca meno conveniente, la *tangente,* gia' si raggiunge un rapporto tra tempo scalare e vettoriale di circa 1.65.

Il massimo Speedup si ottiene con il *Logaritmo* che raggiunge valori vicini a 3.

Frazionamento dell'istruzione in altre ricorrenze

Tipi di ricorrenza

Il metodo del frazionamento dell'istruzione puo' essere applicato ad un qualsiasi tipo di ricorrenza, con l'unica restrizione di porre una particolare attenzione alla memorizzazione del risultato parziale dell'espressione. A tale scopo vengono adesso analizzati altri tipi di ricorrenza e per ognuna di queste e' mostrata la relativa trasformazione.

Indice ausiliario

Un tipo di ricorrenza abbastanza comune e' dovuta all'uso di un indice ausiliario, come e' mostrato nel seguente esempio:

```
      K = 2
      DO  1  I = 1, N
      A(K) = A(I) + B(I) * C(I)
      K = K + 2
1     CONTINUE
```

In questo caso la tecnica del frazionamento del ciclo comporta la scrittura del seguente codice:

```
      K = 2
      DO  1  I = 1, N
      A(K) = B(I) * C(I)
      K = K + 2
1     CONTINUE
      K = 2
      DO  2  I = 1, N
      A(K) = A(I) + A(K)
      K = K + 2
2     CONTINUE
```

Anche in questo caso e' possibile memorizzare i risultati temporanei negli elementi A(K).

Indirizzamento non lineare

Una ricorrenza con indirizzamento non lineare generalemente avviene quando viene usata un'espressione per il calcolo dell'indice. Cio' e' evidenziato nel seguente esempio:

```
      DO  1  I = 1, N
      A(I*I) = A(I) + B(I) * C(I)
1     CONTINUE
```

Anche in questo caso e' possibile riscrivere il ciclo nella forma:

```
      DO  1  I = 1, N
      A(I*I) = B(I) * C(I)
1     CONTINUE
      DO  2  I = 1, N
      A(I*I) = A(I) + A(I*I)
2     CONTINUE
```

Come si nota anche questo tipo di ricorrenza non richiede alcun vettore temporaneo per la memorizzazione dei risultati dell'espressione.

Indirizzamento indiretto

Questo tipo di ricorrenza e' dovuto all'uso degli elementi di un vettore per indirizzare gli elementi di un altro vettore. Un esempio di tale ricorrenza e' il seguente:

```
      DO  1  I = 1, N
      A(INDEX(I)) = A(I) + B(I) * C(I)
1     CONTINUE
```

In questo caso se nel vettore ausiliario *INDEX* esiste una ripetizione di due o piu' valori, allora non e' possibile usare il vettore A per la memorizzazione dei risultati parziali dell'espressione e pertanto e' necessaria la creazione di un vettore temporaneo. Quindi la tecnica del frazionamento dell'istruzione comporta la riscrittura del codice come segue:

```
      DO  1  I = 1, N
      TEMP(I) = B(I) * C(I)
1     CONTINUE
      DO  2  I = 1, N
      A(INDEX(I)) = A(I) + TEMP(I)
2     CONTINUE
```

In questo caso i valori dello Speedup calcolati per le operazioni aritmetiche e per le funzioni intrinseche non sono piu' validi, in

quanto si introducono degli ulteriori accessi alla memoria, a causa dell'introduzione del vettore temporaneo *TEMP*.

E' facile ritenere che il rapporto tra le prestazioni scalari e quelle vettoriali diventi meno favorevole.

Conclusioni

Lo studio condotto sulla possibile eliminazione di una ricorrenza, anche se non ha portato a soluzioni definitive, ha pero' permesso di mettere a fuoco molte problematiche relative al processo di vettorizzazione di un programma.

L'interesse maggiore e' scaturito soprattutto nella fase di prova e di raccolta dei tempi di esecuzione operata, come gia' detto, su di un IBM 3090-VF. Infatti la presenza di certi risultati anomali ha richiesto un approfondimento dell'architettura di tale macchina e il tipo di gestione da parte del sistema operativo MVS della Feature Vettoriale.