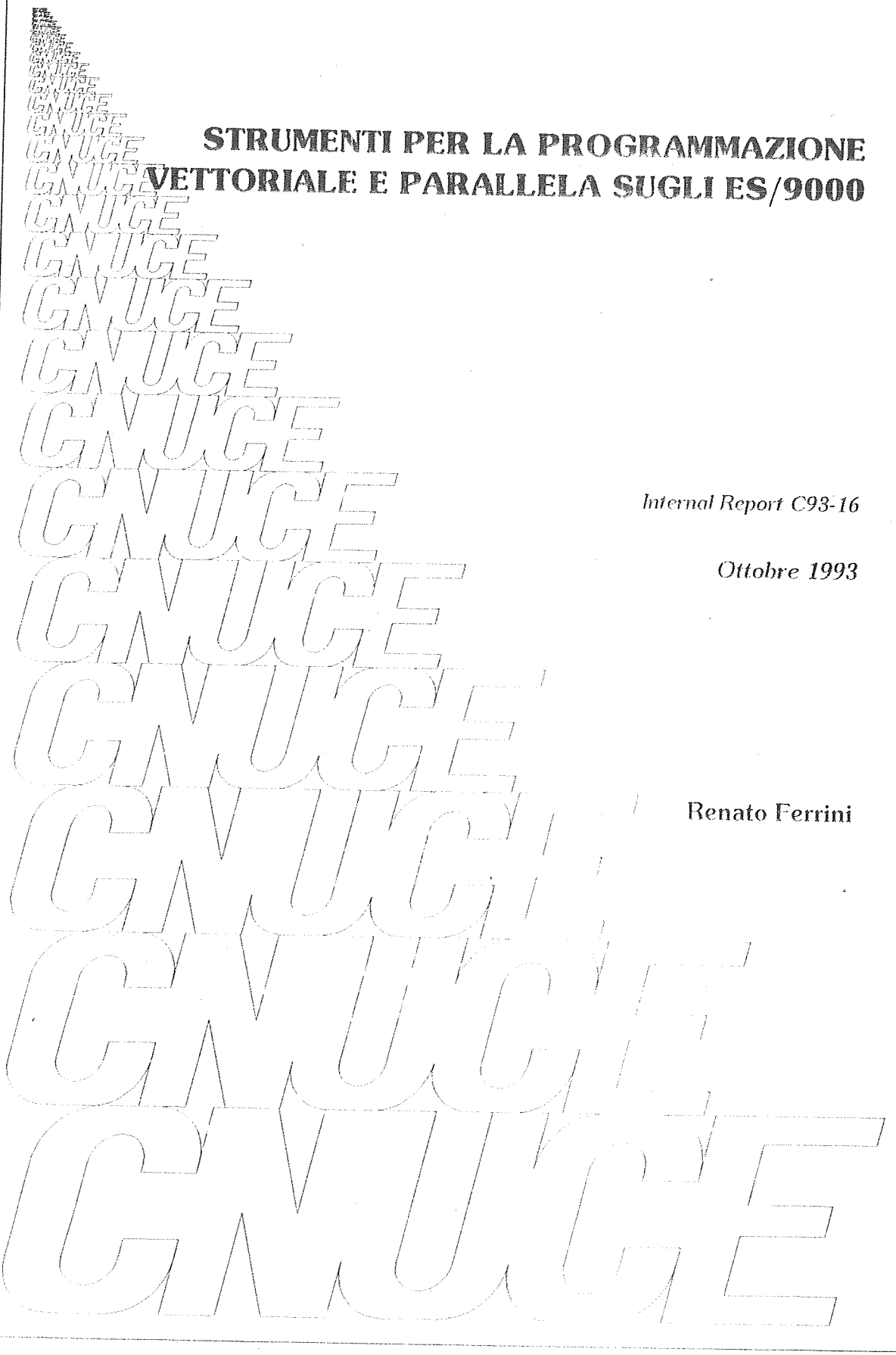
STRUMENTI PER LA PROGRAMMAZIONE VETTORIALE E PARALLELA SUGLI ES/9000



Internal Report C93-16

Ottobre 1993

Renato Ferrini

Strumenti per la programmazione vettoriale e parallela sugli ES/9000

Renato Ferrini

Rapporto Interno C93-16

Pisa, Ottobre 1993

Indice

Introduzione

Nella programmazione vettoriale e parallela assume una notevole importanza il poter disporre di strumenti con cui rilevare le caratteristiche di funzionamento di un'applicazione, sia a livello di intere unità di programma che a livello della singola istruzione. I motivi che determinano questa necessità sono dovuti essenzialmente al fatto che la fase di ottimizzazione di un programma presenta delle difficoltà non indifferenti anche agli stessi addetti ai lavori, perchè troppi e nelle forme più impensate si presentano gli elementi che possono degradare le prestazioni. Una soluzione più semplice consiste invece di disporre di strumenti di utilità in grado di fornire a posteriori, cioè con una prova di esecuzione, il maggior numero di informazioni sulle modalità con cui avviene il processo elaborativo. Come al solito le principali caratteristiche, che è utile conoscere ai fini della valutazione del codice, riguardano essenzialmente l'utilizzo del processore e della memoria e possono essere così riassunte:

- consumo del tempo di calcolo a vari livelli, che possono andare dall'intero programma o alle diverse *subroutine* fino ai cicli *DO* o alle singole istruzioni
- modalità di accesso ai dati, cioè dello *stride* introdotto sui vettori dall'iterazione dei vari indici dei cicli *DO*
- caratteristiche e tempi di utilizzo della memoria *cache* nel tentativo di evitare inutili trasferimenti da e verso la memoria centrale
- sfruttamento dei registri vettoriali al fine di favorire il riutilizzo dei dati

Per quanto riguarda il primo punto, un programmatore esperto potrebbe rilevare i tempi di calcolo usando delle apposite *routine* di sistema, però troverebbe delle notevoli difficoltà per ricavare le informazioni relative agli altri tre punti. Sui sistemi ES/9000 è presente uno strumento *software*, l'*Interactive Debug*, che è in grado di fornire alcune informazioni necessarie per operare una buona ottimizzazione del codice.

Oltre all'*Interactive Debug* esistono ulteriori funzioni di sistema o librerie di programmi che possono venire in aiuto al programmatore. In questo rapporto saranno descritti tutti questi strumenti che sono presenti sugli elaboratori vettoriali e paralleli della serie ES/9000.

5

L' *Interactive Debug*

Le caratteristiche dell'*Interactive Debug*

L'*InterActive Debug* (**IAD**) è un sistema essenzialmente rivolto alla messa a punto dei programmi, ma che può anche essere utilizzato per lo studio delle caratteristiche dell'applicazione e per verificare gli effetti prodotti dalla ottimizzazione e dalla vettorizzazione. Questo strumento viene usato normalmente in maniera interattiva mediante la visualizzazione a terminale di una finestra in cui compare la parte del programma comprendente l'istruzione che è in esecuzione in quel momento.

```
IAD       Q: ESEMPIO                              W: ESEMPIO.3
COMMAND ===>                                        SCROLL ===> PAGE
MONITOR --+----1----+----2----+----3----+----4----+--- LINE 0 OF 0
************************** TOP OF MONITOR **********************
************************** BOTTOM OF MONITOR *******************


SOURCE  0----+----1----+----2----+----3----+----4---- LINE 1 OF 11
       1        PROGRAM ESEMPIO                                    .
       2        INTEGER A(10), B(10), C(10)                        .
       3        DO 1 I=1,10                                        0
       4           A(I) = I+1                                      0
       5           B(I) = I-1                                      0
       6 1      CONTINUE                                           0
       7        CALL DIVIDE (A,B,C)                                0
       8        WRITE(6,2) (C(I),I=1,30)                           0
       9 2      FORMAT (' ',I5)                                    0
      10        STOP                                               0
LOG   0----+----1----+----2----+----3----+----4----+--- LINE 2 OF 4
0002 5668-806 (C) COPYRIGHT IBM CORP 1985, 1988
0004 LICENSED MATERIALS-PROPERTY OF IBM
0005 WHERE: ESEMPIO.3
```

Figura 1 - Esempio di uso dell'Interactive Debug

Un processo di animazione permette poi di modificare costantemente il contenuto della finestra in modo da mostrare l'ultimo

gruppo di istruzioni che è stato interessato dall'elaborazione. Un'altra finestra creata dallo IAD serve per la visualizzazione degli ultimi messaggi stampati dal programma o dall'*Interactive Debug* e anch'essi, come le istruzioni, sono continuamente aggiornati in seguito all'avanzamento dell'esecuzione.

La figura *1* riporta un esempio di *schermata* che compare a terminale quando un programma Fortran viene eseguito sotto il controllo dell'*Interactive Debug*. La situazione rappresentata si riferisce ad un programma Fortran di nome ESEMPIO che è stato appena fatto partire; infatti nell'area di *LOG*, che è identificata dalle ultime 3 righe, è segnalato che l'esecuzione è ferma all'istruzione 3 (messaggio *WHERE* dell'ultima riga), che rappresenta la prima istruzione eseguibile.

Come si può notare lo schermo viene logicamente suddiviso in tre finestre che sono identificate dai nomi *MONITOR*, *SOURCE* e *LOG* e da un campo per l'immissione dei comandi (*COMMAND*). Le zone identificate da *SOURCE* e *LOG* servono, come già detto, a visualizzare rispettivamente il *listing* del programma ed i messaggi dello IAD, mentre la finestra di *MONITOR*, che è attivata da un comando particolare, mostra il contenuto che assumono durante l'esecuzione alcune variabili selezionate.

Per la messa a punto dei programmi l'*Interactive Debug* dispone di una vasta serie di funzioni di cui le principali sono:

- fermare e/o far ripartire il programma da determinate istruzioni
- esaminare e/o modificare il contenuto delle variabili durante l'esecuzione
- alterare il flusso logico dell'esecuzione
- manipolare i *files* esterni
- modificare la dimensione ed i colori delle finestre che compaiono su video
- selezionare le unità di programma che sono interessate dall'esecuzione controllata
- disporre di informazioni sul funzionamento dello IAD e sui formati dei comandi direttamente in linea
- far ripartire la sessione dall'inizio

La distribuzione dei tempi di calcolo

Ma le funzioni dell'*Interactive Debug* che hanno una maggiore interesse ai fini della vettorizzazione del programma sono quelle che permettono di conoscere la distribuzione del tempo di calcolo tra le varie unità di programma, o tra le varie istruzioni, e di analizzare alcune caratteristiche dei cicli *DO*. Il principio con cui viene svolta la prima funzione si basa sul campionamento del tempo di calcolo mediante delle interruzioni ad intervalli regolari dell'esecuzione e sulla registrazione dell'istruzione, e della relativa unità di programma, trovate attive al momento dell'interruzione. L'operazione di campionamento viene innescata con il comando *ENDDEBUG SAMPLE* che va emesso all'interno della finestra dello IAD sulla riga di *COMMAND*; un parametro aggiuntivo del comando *ENDDEBUG* permette anche di specificare l'intervallo del campionamento che, se omesso, è pari a 10 millisecondi.

```
0001    PROGRAM CALCOLO
0002      REAL*4 A(100,100),B(100,100),C(100,100)
0003      COMMON /UNO/ A, B, C
0004      DO  1  I=1,100
0005         DO  1  J=1,100
0006            A(I,J)=I
0007            B(I,J)=J
0008            C(I,J)=0.
0009    1 CONTINUE
0010      CALL PROD
0011      STOP
0012      END

0001      SUBROUTINE PROD
0002      REAL*4 A(100,100),B(100,100),C(100,100)
0003      COMMON /UNO/ A, B, C
0004      DO  1  I=1,100
0005         DO  1  J=1,100
0006            DO  1  K=1,100
0007               C(I,J)=C(I,J)+A(I,K)*B(K,J)
0008    1 CONTINUE
0009      RETURN
0010      END
```

Figura 2 - Codice Fortran per il prodotto tra 2 matrici

Un esempio del funzionamento dello IAD, per quanto riguarda la distribuzione del tempo di calcolo, viene adesso analizzato con il programma di figura *2*, che è composto da un'unità principale di nome *CALCOLO* e da una subroutine *PROD* che svolge il prodotto tra due matrici e che quindi presumibilmente consumerà la maggior parte del tempo di esecuzione. Prima di usare l'*Interactive Debug* il programma deve essere compilato con il compilatore Fortran VS, specificando anche l'opzione *SDUMP* nel caso che si desideri la distribuzione del tempo di calcolo a livello della singola istruzione. La fase successiva consiste nel richiamare l'*Interactive Debug* e di procedere al campionamento del programma usando il comando *ENDDEBUG SAMPLE.*

```
listsamp * summary

PROGRAM SAMPLING INTERVAL WAS 10 MS; TOTAL NUMBER OF SAMPLE WAS 30.
DIRECT SAMPLE:
PROGRAM UNIT                          SAMPLES  %TOTAL
CALCOLO                                     1   3.33 *
PROD                                       29  96.67 ********************
```

Figura 3 - Distribuzione del tempo di calcolo del programma di figura 2

Una volta raccolti i dati relativi alla distribuzione del tempo di calcolo, l'*Interactive Debug* consente la visualizzazione di vari tipi di elaborati statistici ottenuti con il comando *LISTSAMP*; il primo risultato, che viene mostrato in figura *3*, fornisce le informazioni a livello di unità di programma e quindi a carattere generale.

```
listsamp prod.*

PROGRAM SAMPLING INTERVAL WAS 10 MS; TOTAL NUMBER OF SAMPLE WAS 30.
DIRECT SAMPLE:
STATEMENT                  SAMPLES  %UNIT %TOTAL
PROD.ENTRY/EXIT                  0   0.00   0.00
PROD.4                           0   0.00   0.00
PROD.5                           0   0.00   0.00
PROD.6                           4  13.79  13.33 ***
PROD.7                          24  82.76  80.00 ****************
PROD.8/1                         1   3.45   3.33 *
PROD.9                           0   0.00   0.00
PROD.10                          0   0.00   0.00
```

Figura 4 - Distribuzione del tempo di calcolo della subroutine PROD

I risultati statistici confermano le previsioni che indicavano le operazioni presenti nella subroutine *PROD* come le più complesse dell'intero programma e quindi quelle che richiedono una maggiore quantità di tempo di calcolo. Tra le informazioni fornite dal comando *LISTSAMP* ci sono anche la durata del tempo di campionamento e il numero di campionamenti effettuati che in questo caso è pari a 30. Da questi due parametri è anche possibile risalire approssimativamente al tempo totale di esecuzione dell'applicazione che per il programma di figura *2* si dovrebbe aggirare sui 300 millisecondi.

Un maggior dettaglio dei tempi di esecuzione si ottiene scendendo a livello delle singole istruzioni. Questa indagine può essere condotta per tutte le istruzioni del programma, oppure essere rivolta solo alle unità di programma dove si addensa la maggior quantità del tempo di calcolo. Poichè nell'esempio di figura *2* il nostro interesse è quello di capire quali sono i motivi che nella routine *PROD* comportano un uso elevato del processore, sarà adottata la seconda soluzione.

Le informazioni a livello di istruzione riportate in figura *4* indicano che i motivi del consumo del tempo di calcolo da parte della subroutine *PROD* sono da imputare sia alle operazioni di somma e prodotto presenti nell'istruzione *0007*, dove viene speso l'80% del tempo di esecuzione, che alle istruzioni di controllo del ciclo più interno le quali richiedono complessivamente una percentuale del tempo di esecuzione pari al 16.67%. Questi dati statistici potevano anche essere disposti direttamente nel *listing* del programma per una migliore lettura dei risultati, come mostra la figura *5*.

```
annotate prod

IF DO  ISN                                                    %TOTAL DISTRIBUTION

       1   SUBROUTINE PROD                                CAL00160
       2   REAL*4 A(100,100),B(100,100),C(100,100)        CAL00170
       3   COMMON /UNO/ A, B, C                           CAL00180
       4   DO  1  I=1,100                                 CAL00190
   1   5      DO  1  J=1,100                              CAL00200
   2   6         DO  1  K=1,100                           CAL00210  13.33 ***
   3   7            C(I,J)=C(I,J)+A(I,K)*B(K,J)           CAL00220  80.00 ****************
   3   8 1 CONTINUE                                       CAL00230   3.33 *
       9   RETURN                                         CAL00240
      10   END                                            CAL00250
```

Figura 5 - Distribuzione del tempo di calcolo della subroutine PROD

Il comando *ANNOTATE* svolge la funzione di inserire i risultati statistici direttamente nel *listing* del programma a fianco

dell'istruzione interessata. Questa operazione risulta molto comoda per le unità di programma che sono composte da migliaia di righe di codice.

Il controllo della vettorizzazione

Un'altra possibilità offerta dall'*Interactive Debug* è quella di valutare gli effetti della vettorizzazione e quindi di capire le scelte operate dal compilatore. Per questo tipo di operazioni è necessario che il programma sia stato compilato usando il parametro *IVA* dell'opzione *VECTOR*. Un primo elemento che è possibile ricavare sono i tempi di calcolo ottenuti dopo la vettorizzazione sia a livello delle varie unità di programma che a livello dei cicli *DO*. I comandi dell'*Interactive Debug* preposti a tale scopo sono *TIMER* e *LISTTIME*, il cui effetto sul programma di figura *2* produce i risultati di figura *6*.

```
listtime

ENTRY POINT                TOTAL TIME  %TOTAL  INVOCATIONS  AVERAGE TIME
CALCOLO                          1428    1.71            1          1428
PROD                            83038   98.29            1         83038

listtime prod.* doloop

DO LOOP                    TOTAL TIME  EXECUTIONS  AVERAGE TIME  STATUS
PROD.4                          83821           1         83821    ON
PROD.5                          83700           1         83700    ON
PROD.6                          79393         100           794    ON
```

Figura 6 - Stampa dei tempi dell'esecuzione vettoriale

Il tempi forniti dall'*Interactive Debug* sono espressi in microsecondi e rappresentano il tempo totale di esecuzione non comprensivo dell'*overhead* introdotto dal sistema operativo e dalle eventuali operazioni di paginazione. Il suddetto tempo risente invece degli ulteriori ritardi dovuti alla gestione del programma da parte dello *IAD*; pertanto la sua misura non è molto precisa e può diventare del tutto inattendibile se la durata totale dell'applicazione è molto ridotta, come nel nostro esempio. Nonostante ciò l'indicazione sui tempi rappresenta un'informazione sempre utile per valutare gli effetti dell'ottimizzazione o vettorizzazione del programma.

I risultati mostrati in figura *6* sono stati ottenuti con l'uso del comando *LISTTIME*, ma è bene precisare che tale comando si

limita solamente a visualizzare a terminale un insieme di dati che è stato precedentemente raccolto con una esecuzione del programma sotto l'effetto del comando *TIMER*. Il primo risultato prodotto dall'*Interactive Debug* fornisce alcune informazioni a livello di unità di programma, comprendenti il tempo di esecuzione di ciascuna *routine*, la sua percentuale rispetto alla durata totale del programma, il numero di volte che ogni *routine* è stata richiamata e la sua durata media. Un elemento interessante che scaturisce da questi dati è la riduzione del tempo di calcolo che è avvenuta nel passaggio dall'esecuzione scalare, che era di circa 300 *ms*, a quella vettoriale che è di circa 84.5 *ms*. L'informazione relativa alla durata di ciascuna unità di programma poteva essere ricavata anche per l'esecuzione scalare e ciò allo scopo di valutare in maniera più accurata i vantaggi ottenuti con l'uso della *Vector Facility*.

Il secondo gruppo di informazioni statistiche presenti in figura *6* si riferisce invece ad alcune caratteristiche dei cicli *DO* appartenenti alla *subroutine* PROD. Il tempo totale di ciascun ciclo comprende anche la durata di eventuali cicli più interni, per cui la differenza tra il ciclo dell'istruzione *0005* e quello dell'istruzione *0006*, che supera di poco i 4 msec, è dovuta essenzialmente alle istruzioni di controllo che sono necessarie per operare le 100 iterazioni. La stessa conclusione vale anche per spiegare la differenza di 0,1 msec esistente tra i cicli delle istruzioni *0004* e *0005*.

L'elemento, invece, che ad una prima analisi può sembrare strano è il numero di esecuzioni del ciclo dell'istruzione *0005* che per effetto delle iterazioni imposte dall'istruzione *0004* dovrebbe essere eseguito 100 volte. Questo fenomeno è conseguenza degli effetti della vettorizzazione operata sull'indice *I*, per i quali il ciclo dell'istruzione *0005* viene eseguito un numero di volte pari al numero delle sezioni dei vettori da elaborare, che in questo caso è 1.

Informazioni più dettagliate sui cicli *DO* vettorizzati possono essere fornite dall'*Interactive Debug* con i comandi *VECSTAT* e *LISTVEC*. Come nel caso dei comandi *TIMER* e *LISTTIME*, il comando *VECSTAT* serve per raccogliere i dati durante un'esecuzione controllata del programma ed il comando *LISTVEC* per ridurre tali dati e visualizzare le informazioni così ottenute.

In figura *7* sono stati riportati i risultati che si ricavano usando i comandi *VECSTAT* e *LISTVEC* con il programma di figura *2*. Come si può notare, per ogni ciclo *DO* della *subroutine* PROD vengono forniti, oltre al numero di volte che è stato eseguito, anche il numero medio ed il numero stimato delle iterazioni dell'indice del ciclo. Poichè in questo caso il limite del ciclo è espresso mediante una costante non esiste alcuna differenza tra i

due valori.

Un'altra informazione importante fornita dal comando *LISTVEC* si riferisce allo *stride* introdotto dall'uso dell'indice sui vettori presenti nelle istruzioni interne al ciclo. Per quanto riguarda il ciclo *DO* dell'istruzione *0004* le informazioni visualizzate dallo IAD indicano che sono interessate dall'indice *I* le matrici *A* e *C* nell'istruzione *0007* ed il riferimento ai loro elementi avviene con uno *stride* pari a 1. Questo è il motivo per cui questo indice viene preferito agli altri due per la vettorizzazione dell'istruzione *0007*. Infatti il ciclo con indice *J* provoca un accesso agli elementi delle matrici *B* e *C* con uno *stride* di 100 e lo stesso effetto è prodotto dall'indice *K* sulla matrice *A*. Quest'ultimo risultato riguardante l'indice *K* era già stato evidenziato nel capitolo sull'architettura degli elaboratori ES/9000 a riguardo dell'ottimizzazione del prodotto tra matrici sulla *Vector Facility* ed costituiva la causa per cui veniva scartata la vettorizzazione su tale indice anche se portava il vantaggio dell'uso di un'istruzione di accumulazione.

```
listvec prod.*

PROD.4:
  STATUS = ON
  TOTAL NUMBER OF EXECUTION = 1
  AVERAGE ITERATION COUNT = 100
  ESTIMATED ITERATION COUNT = 100
    STATEMENT              ARRAY              AVG STRIDE  EST STRIDE
  V PROD.7                 A                           1           1
  V PROD.7                 C                           1           1
  V PROD.7                 C                           1           1
PROD.5:
  STATUS = ON
  TOTAL NUMBER OF EXECUTION = 1
  AVERAGE ITERATION COUNT = 100
  ESTIMATED ITERATION COUNT = 100
    STATEMENT              ARRAY              AVG STRIDE  EST STRIDE
  S PROD.7                 B                         100         100
  S PROD.7                 C                         100         100
  S PROD.7                 C                         100         100
PROD.6:
  STATUS = ON
  TOTAL NUMBER OF EXECUTION = 100
  AVERAGE ITERATION COUNT = 100
  ESTIMATED ITERATION COUNT = 100
    STATEMENT              ARRAY              AVG STRIDE  EST STRIDE
  S PROD.7                 A                         100         100
  S PROD.7                 B                           1           1
```

Figura 7 - Caratteristiche dei cicli DO della subroutine PROD

Poichè quasi tutte le informazioni ottenute con i comandi *VECSTAT* e *LISTVEC* dell'*Interactive Debug* sono fornite dal compilatore Fortran VS con l'opzione *VECTOR(REPORT(STAT))*, qualcuno potrebbe chiedersi qual è la differenza tra queste due possibilità. Ebbene il vantaggio della prima rispetto alla seconda

è che, mentre il compilatore è costretto in molti casi a supporre il valore di alcune informazioni perchè non è in grado di conoscere il contenuto che assumeranno le variabili interessate in fase di esecuzione, l'I*nteractive Debug* svolge le stesse indagini proprio durante l'esecuzione e quindi fornisce delle indicazioni più attendibili.

```
0001    PROGRAM CALCOLO
0002      REAL*4 A(100,100),B(100,100),C(100,100)
0003      DO  1  I=1,100
0004         DO  1  J=1,100
0005            A(I,J)=I
0006            B(I,J)=J
0007            C(I,J)=0.
0008    1 CONTINUE
0009      CALL PROD(A,B,C,100)
0010      STOP
0011      END

0001      SUBROUTINE PROD(A,B,C,N)
0002      REAL*4 A(N,N),B(N,N),C(N,N)
0003      DO  1  I=1,N
0004         DO  1  J=1,N
0005            DO  1  K=1,N
0006               C(I,J)=C(I,J)+A(I,K)*B(K,J)
0007    1 CONTINUE
0008      RETURN
0009      END
```

Figura 8 - Codice Fortran per il prodotto tra 2 matrici

A riprova di questa affermazione modifichiamo il programma di figura *2* per ottenere il codice riportato in figura *8*.

```
                    VECTOR STATISTICS TABLE

ISN  ARRAY/INDUCTION        LEVEL 1       LEVEL 2       LEVEL 3

  3  I                     COUNT=65?
  4  J                                   COUNT=65?
  5  K                                                 COUNT=65?
  6  A                           1V                          65?
     C                           1V            65?
     C                           1V            65?
     B                                         65?            1
```

Figura 9 - Esempio di compilazione con il parametro REPORT(STAT)

A differenza del programma originale in cui nella *subroutine PROD* le dimensioni delle matrici *A*, *B* e *C* erano note, nel codice di figura *8* le matrici e la loro dimensione sono trasmessi alla *subroutine PROD* solo al momento della chiamata.

Un programma così strutturato produce la lista di informazioni riportata in figura *9* quando viene compilato con l'opzione *VECTOR(REPORT(STAT))*. Come si può notare il compilatore è costretto a supporre sia il numero delle iterazioni degli indici *I*, *J* e *K*, che lo *stride* prodotto dai livelli 2 e 3 sulle matrici *A*, *B* e *C*. Se invece il programma di figura *8* è eseguito sotto il controllo dell'*Interactive Debug*, utilizzando i comandi *VECSTAT* e *LISTVEC*, si ottiene una lista di informazioni come quella riportata in figura *10*.

```
listvec prod.*

PROD.4:
  STATUS = ON
  TOTAL NUMBER OF EXECUTION = 1
  AVERAGE ITERATION COUNT = 100
  ESTIMATED ITERATION COUNT = 65?
    STATEMENT                ARRAY                  AVG STRIDE  EST STRIDE
  V PROD.6                   A                               1           1
  V PROD.6                   C                               1           1
  V PROD.6                   C                               1           1
PROD.5:
  STATUS = ON
  TOTAL NUMBER OF EXECUTION = 1
  AVERAGE ITERATION COUNT = 100
  ESTIMATED ITERATION COUNT = 65?
    STATEMENT                ARRAY                  AVG STRIDE  EST STRIDE
  S PROD.6                   B                             100         65?
  S PROD.6                   C                             100         65?
  S PROD.6                   C                             100         65?
PROD.6:
  STATUS = ON
  TOTAL NUMBER OF EXECUTION = 100
  AVERAGE ITERATION COUNT = 100
  ESTIMATED ITERATION COUNT = 65?
    STATEMENT                ARRAY                  AVG STRIDE  EST STRIDE
  S PROD.6                   A                             100         65?
  S PROD.6                   B                               1           1
```

Figura 10 - Caratteristiche dei cicli DO della subroutine PROD

Come evidenzia la figura *10*, accanto ai valori stimati compaiono anche i valori che l'*Interactive Debug* ha registrato durante l'esecuzione e che forniscono una indicazione più precisa ai fini di una migliore ottimizzazione del programma.

15

Le funzioni intriseche

Le funzioni vettorizzate

Alcune funzioni intrinseche di libreria sono disponibili in una versione vettoriale che, oltre a migliorare le prestazioni dell'applicazione, non rappresenta più un inibitore alla vettorizzazione del ciclo. Nella libreria Fortran VS versione 2 i risultati ottenuti dalla versione vettoriale sono identici a quelli della versione scalare della stessa funzione, ma possono essere diversi dalla versione scalare della libreria Fortran VS versione 1.

Funzioni in linea

SQRT	DLOG	DTAN	CDABS
DSQRT	SIN	DCOTAN	ALOG10
EXP	DSIN	ATAN	DLOG10
DEXP	COS	DATAN	ATAN2

Funzioni fuori linea

HFIX	IFIX	COMPLX	SNGL
IABS	INT	DCMPLX	DBLE
ABS	IOR	CONJG	AMAX1
DABS	ISHFT	DCONJG	DMAX1
IAND	NOT	FLOAT	MAX1
IBCLR	AIMAG	DFLOAT	AMIN1
IBSET	DIMAG	DPROD	DMIN1
IDENT	AINT	REAL	MIN1
IEOR	DINT	DREAL	

Tabella 11 - Lista delle funzioni vettoriali di libreria

La versione vettoriale di una funzione viene attivata dal compilatore quando è stato specificato il parametro *INTRINSIC* dell'opzione *VECTOR.*

Funzioni fuori linea

ACOS	ERF	DTANH	CDLOG
DACOS	DERF	TAN	CSQRT
ASIN	ERFC	CCOS	CDSQRT
DASIN	DERFC	CDCOS	IBCLR
COTAM	GAMMA	CSIN	IBSET
COSH	DGAMMA	CDSIN	ISHFT
DCOSH	ALGAMMA	CEXP	
SINH	DLGAMMA	CDEXP	
DSINH	TANH	CLOG	

Tabella 12 - Lista delle funzioni scalari di libreria con argomenti vettoriali

Le funzioni di libreria presenti in figura *11* sono disponibili sia in versione scalare che vettoriale e sono state suddivise tra quelle per cui il compilatore genera un codice direttamente all'interno del programma (*in linea*) e quelle che sono risolte con una chiamata ad una *routine* esterna (*fuori linea*).

Esiste poi un gruppo di funzioni di libreria che sono eseguite in modo scalare ma accettano in entrata e forniscono in uscita, argomenti vettoriali e pertanto non inibiscono la vettorizzazione del ciclo. In figura *12* è riportata una lista di queste funzioni, mentre tutte le altre funzioni di libreria non presenti nelle tabelle *11* e *12* sono eseguite in modo scalare.

17

La libreria ESSL

La *Engineering and Scientific Subroutine Library*

Durante la trattazione delle modalità di implementazione di una applicazione vettoriale è stato fatto riferimento alla possibilità di svolgere parte o tutte le funzioni previste nell'algoritmo usando dei sottoprogrammi già disponibili nel sistema di lavoro. Il *software* che ottempera a questa funzione sugli ES/9000 è la libreria di sottoprogrammi matematici *Engineering and Scientific Subroutine Library* (**ESSL**). La libreria ESSL è essenzialmente preposta all'espletamento di funzioni che sono di uso comune in molte aree scientifiche ed ingegneristiche, come l'esplorazione sismica, la ricerca geofisica, l'elaborazione di immagini, ecc... e che sono caratterizzate da una grande quantità di dati e da una elevato numero di calcoli da svolgere. In pratica si è cercato di dare una soluzione a tutti quei problemi connessi all'implementazione di un codice o di difficile scrittura, perchè richiede conoscenze matematiche ed informatiche all'avanguardia, o per il quale il compilatore non è in grado di fornire un codice eseguibile che abbia delle buone prestazioni.

La libreria ESSL è composta da una serie di sottoprogrammi, disponibili sia in versione scalare che vettoriale, richiamabili da programmi Fortran compilati con il Fortran VS versione 2 o da programmi *assembler* compilati con l'Assembler H versione 2. La versione scalare dei sottoprogrammi è stata introdotta in vista di un suo utilizzo durante la fase di messa a punto dell'applicazione e per valutare l'incremento delle prestazioni nel passaggio da scalare a vettoriale, mentre è sconsigliato un suo uso a regime in quanto, trattandosi di operazioni tra vettori o matrici, la versione vettoriale è sempre più conveniente di quella scalare. Le *subroutine* della libreria ESSL si basano su algoritmi che offrono una elevata precisione dei risultati e che utilizzano in maniera più efficiente possibile l'architettura scalare e vettoriale degli elaboratori ES/9000 mediante tecniche che hanno l'obiettivo di:

- conservare il più possibile i dati nella memoria *cache*
- accedere ai dati con *stride* uguale a 1
- mantenere i dati nei registri vettoriali e/o scalari
- favorire il massimo tasso di riutilizzo dei dati
- utilizzare le istruzioni di macchina più efficienti
- minimizzare le operazioni di paginazione

I sottoprogrammi, che possono operare con dati espressi sia in singola (32 *bit*) che in doppia precisione (64 *bit*), svolgono funzioni matematiche inerenti i seguenti campi:

1. *Algebra lineare*
   Si tratta di sottoprogrammi che rappresentano un sottoinsieme del codice presente nella *Basic Linear Algebra Subprograms* (BLAS) e che eseguono operazioni tra scalari e vettori, tra vettori e vettori e tra vettori e matrici. Un sottoinsieme di queste *routine* è predisposta a svolgere le suddette operazioni nel caso di matrici o vettori sparsi.

2. *Equazioni di algebra lineare*
   Questo insieme di sottoprogrammi provvede alla risoluzione di sistemi lineari di equazioni con matrici generali reali o complesse, o con matrici generali a banda, o con matrici reali simmetriche definite positive o con matrici simmetriche definite positive a banda.

3. *Autovalori e autovettori*
   Questa serie di sottoprogrammi calcolano gli autovalori e gli autovettori di matrici generali reali o complesse o di matrici reali simmetriche.

4. *Trattamento dei segnali*
   Sono sottoprogrammi che svolgono operazioni classiche di analisi di frequenza o armonica come la trasformata di Fourier, la correlazione, la convoluzione, ecc...

5. *Generazione di numeri casuali*
   A questa funzione sono preposti due sottoprogrammi, uno operante in singola e l'altro in doppia precisione, che forniscono a partire da un valore iniziale un vettore di numeri

casuali.

6. *Funzioni di utilità*
   Sono un insieme di sottoprogrammi che svolgono varie funzioni di utilità come l'ordinamento di un vettore o la conversione di una matrice sparsa in una matrice compressa, ecc...

Per le sue caratteristiche di utilizzo ottimale degli elementi architetturali degli ES/9000, il ricorso ai sottoprogrammi della libreria ESSL offre le migliori prestazioni che si possono ottenere per realizzare una determinata funzione. A tal fine possiamo far riferimento al prodotto tra due matrici definite in doppia precisione presentato nel capitolo relativo all'architettura della *Vector Facility* dove, a seguito di successive ottimizzazioni, si raggiungevano su di un modello avente un ciclo di macchina di 14.5 nsec delle prestazioni di 58 *MFLOPS*. Se tale operazione viene eseguita da un sottoprogramma della libreria ESSL su di un elaboratore con le stesse caratteristiche si ottengono delle prestazioni di poco superiori a 76 *MFLOPS*.

Questi risultati consigliano pertanto di usare le *subroutine* della libreria ESSL in tutte le zone del programma dove si spende la maggior parte del tempo di calcolo; in questo modo si ha il vantaggio di risolvere facilmente la fase di vettorizzazione del codice e di ottenere immediatamente delle ottime prestazioni.

Gli strumenti di aiuto alla vettorizzazione disponibili sui sistemi ES/9000 rappresentano un pezzo delle possibilità offerte dalla IBM per il calcolo vettoriale, che vanno dalle componenti *hardware* adibite al calcolo intensivo fino a tutti i pacchetti *software* che direttamente o indirettamente sono coinvolti nel processo di vettorizzazione. La ricchezza e la varietà di strumenti presenti anche su altri elaboratori e gli sforzi fatti dai costruttori per rendere la programmazione vettoriale sempre più automatizzata fanno ritenere ormai tramontata l'era dell'utente costretto a diventare anche un esperto di informatica e lasciano presagire un sempre maggiore impiego del calcolo vettoriale e parallelo nei più svariati settori applicativi.

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

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