coordinamento di Andrea de Prisco

Data Flow Computers

di Giuseppe Cardinale Ciccotti

Nella ricerca di performance sempre maggiori, sono stati esplorati nuovi approcci nel progetto degli elaboratori; uno di questi, fra i più significativi, è sicuramente l'architettura «data-flow». Il concetto alla base di tale tipo di architettura propone un nuovo modo di considerare l'elaborazione, tanto originale da creare una categoria a sé stante e viene in genere indicato come «data-driven», in contrapposizione all'elaborazione tradizionale di tipo «control-driven». Lo stesso concetto è alla base dei linguaggi funzionali, quali per esempio il Lisp; è possibile quindi evitare i problemi tipici di conversione seriale-parallelo che affliggono tutti gli altri tipi di elaborazione parallela. Cercheremo, in questo appuntamento, di evidenziare i concetti di base e le architetture che ne derivano, di questo tipo di elaborazione alternativo

Computazione data-driven e control-driven

I computer data-flow sono basati sul concetto di computazione «data-driven», che è drasticamente differente dal concetto di elaborazione alla base di una convenzionale macchina di tipo Von Neumann. La differenza sostanziale è che l'esecuzione delle istruzioni di un computer tradizionale, è controllata dal flusso del programma, mentre in un computer data-flow è guidata dalla disponibilità dei dati. Le differenze tra i concetti di control-flow e data-flow possono essere ben esplicitate da un diverso modo di rappresentare il flusso dei programmi. In figura 1, è riportata la rappresentazione, seriale e parallela, di un frammento di programma per il calcolo dell'espressione a= (b+1) * (b−c) che come si vede, è specificato da una serie di istruzioni ed un esplicito controllo del flusso. Notate che i dati vengono condivisi tra le istruzioni accedendo alle medesime locazioni di memoria; le variabili in particolare, sono referenziate attraverso l'indirizzo della locazione di memoria in cui sono contenute. Nella figura 1 inoltre, abbiamo scelto di rappresentare con archi pieni l'accesso ai dati e il flusso di controllo.

Nel tradizionale modello «control-flow sequenziale» (Von Neumann), c'è un singolo flusso di controllo, come potete notare in figura 1a, che è passato da un'istruzione all'altra. L'esecuzione di istruzioni come GOTO, FOR o WHILE provocano espliciti trasferimenti di controllo. Nel modello «control-flow parallelo», in figura 1b, speciali costrutti, quali FORK e JOIN, sono usati per controllare il parallelismo. Tali operatori permettono

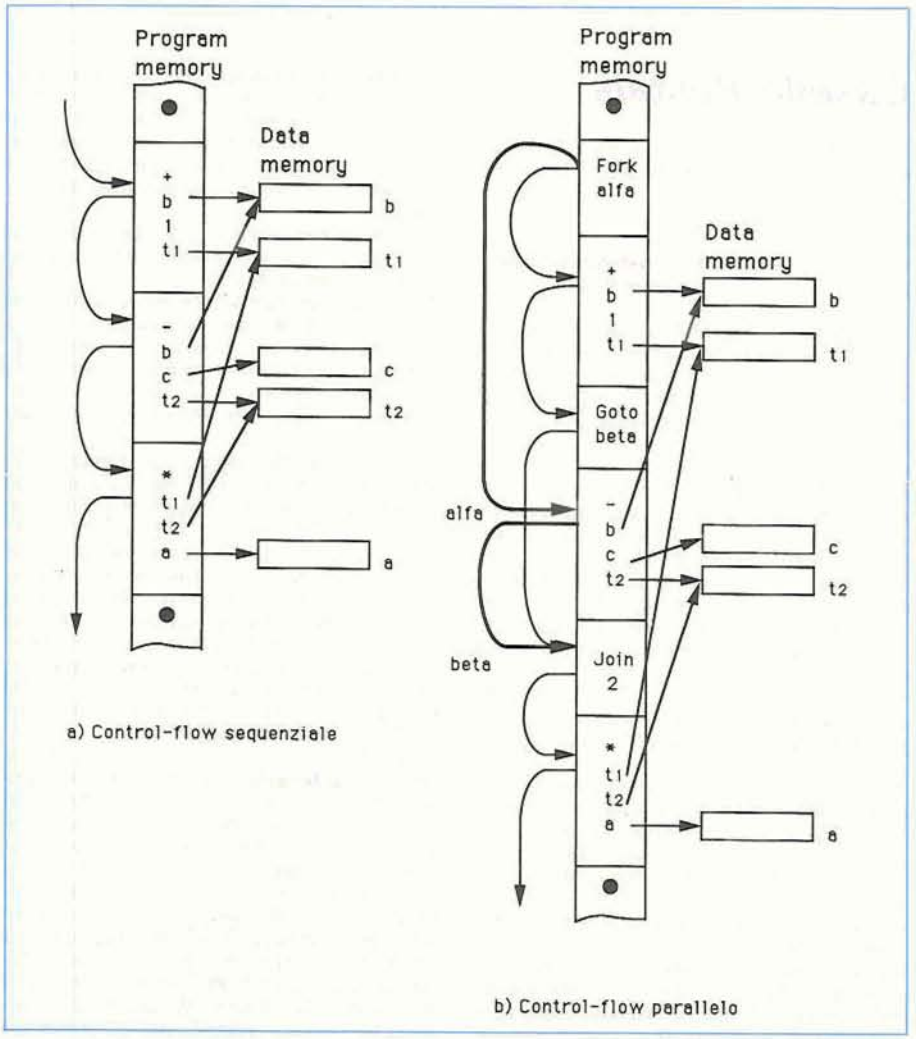


Figura 1 - Sequenza delle istruzioni in un computer "control-flow" per il calcolo della espressione a=(b+1)(b-c).*

di attivare e di sincronizzare più flussi di controllo contemporaneamente. Il modello «control-flow» può essere quindi sintetizzato nelle seguenti caratteristiche:
— i dati tra le istruzioni sono passati tramite l'indirizzo della cella di memoria condivisa.
— Il flusso di controllo è implicita sequenziale, ma speciali operatori di controllo possono essere usati per generare e sincronizzare il parallelismo.
— Sono presenti registri «Program counters» per sequenziare l'esecuzione di istruzioni in un ambiente di controllo centralizzato.

In un modello «data-flow» le istruzioni sono attivate dalla disponibilità dei dati indicati delle parentesi () in figura 2. I programmi data-flow sono rappresentati da grafi diretti, in cui gli archi che connettono i nodi sono orientati in una sola direzione, che rappresenta il flusso dei dati tra le istruzioni. Ciascuna istruzione consiste di un operatore, uno o due operandi e una o più destinazioni al quale il risultato deve essere mandato. In figura 2 potete ritrovare il medesimo frammento di codice della figura 1 rappresentato secondo il modello data-flow, mentre in figura 3 sono presenti tre «istantanee» del flusso dei dati durante l'esecuzione del programma stesso in ambiente data-flow. I pallini corrispondono ai dati passati tra le istruzioni. Ricapitoliamo quindi, le caratteristiche salienti del modello data-flow:
— i risultati sia intermedi che finali, sono passati direttamente tra le istruzioni.
— Non esiste il concetto di memorizzazione del dato, insito nella tradizionale nozione di variabile.
— La sequenza del programma è vincolata soltanto dalla dipendenza dei dati tra le istruzioni.

Dopo queste precisazioni si inizia a comprendere come gli elaboratori control-flow abbiano una organizzazione control-driven; il programma ha infatti, il controllo totale della sequenza delle istruzioni. Per contro, i computer data-flow, seguono il concetto di data-driven che si risolve nell'esame delle istruzioni per verificare la disponibilità degli operandi e quindi ordinarne l'esecuzione non appena l'unità funzionale interessata (per esempio un addizionatore se l'istruzione è un'addizione), sia libera. Questo modo di considerare l'elaborazione comporta intrinsecamente l'esecuzione asincrona e simultanea delle istruzioni requisiti auspicabile il primo e

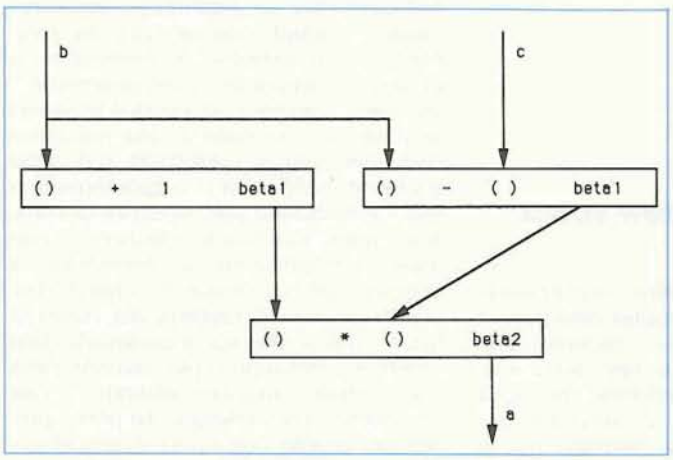


Figura 2 - Esecuzione delle istruzioni in un computer data-flow per il calcolo dell'espressione a=(b+1)(b-c).*

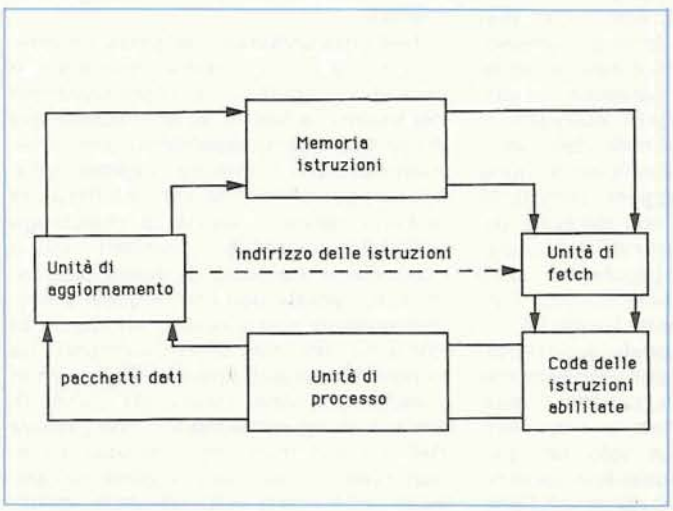


Figura 4 - Schema funzionale di un computer ed architettura data-flow statica.

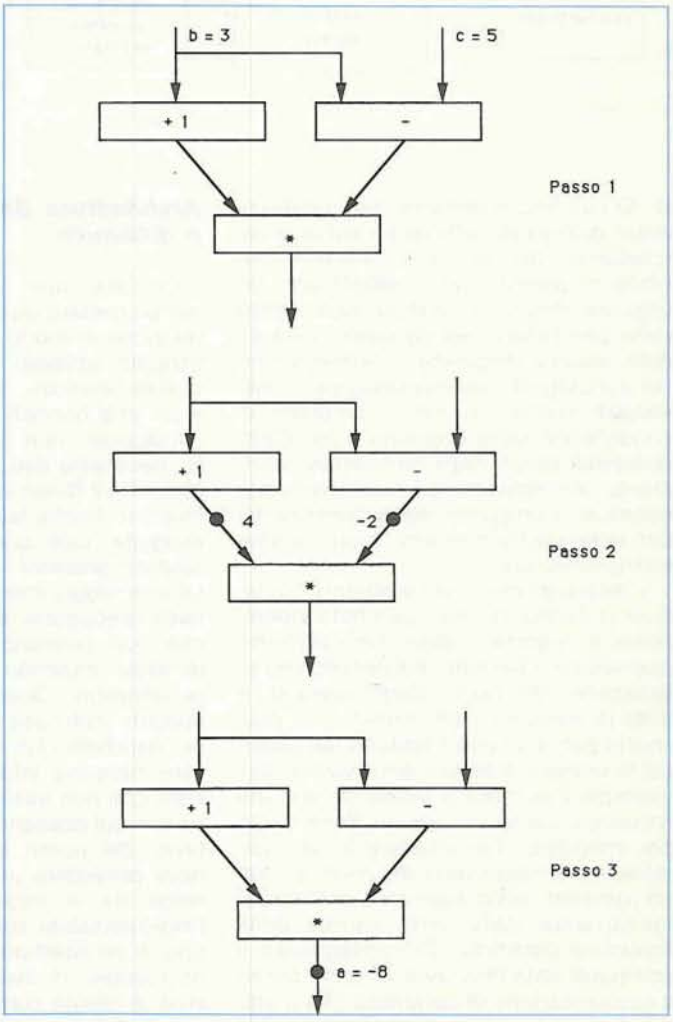


Figura 3 - I tre passi eseguiti da un computer data-flow per il calcolo a=(b+1)(b-c). I punti sugli arc indicano che il dato è presente e ha il valore indicato.*

indispensabile il secondo, per un efficiente parallelismo. La qualità più notevole sta, comunque, nel fatto che tutto il parallelismo ottenuto è implicito, nel senso che non deve essere espresso direttamente dal programmatore, al contrario di quanto osservato nelle architetture precedentemente considera-

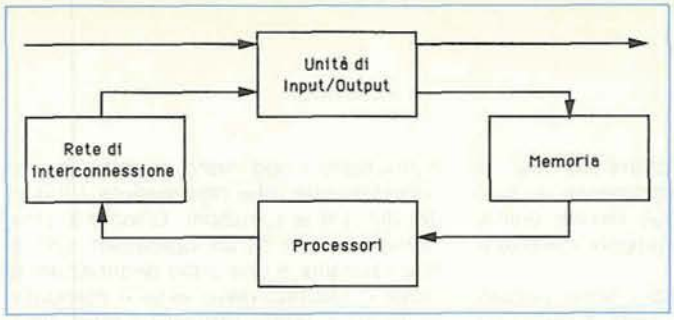


Figura 6 - Struttura generale data-flow ad anello. La sezione di input/output esegue il controllo sui dati.

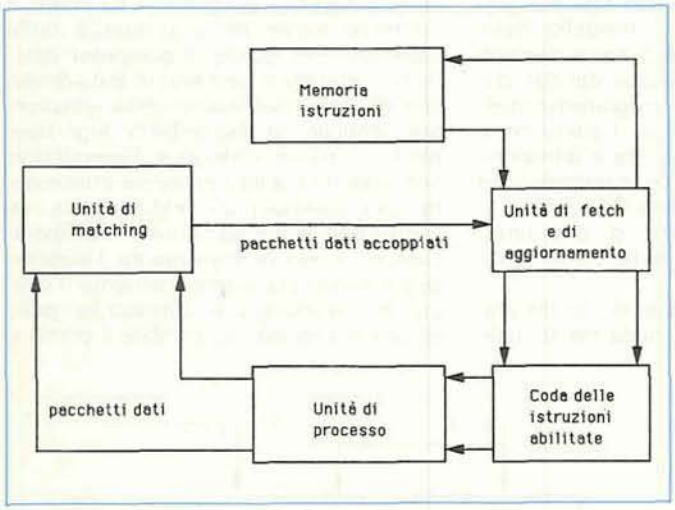


Figura 5 - Schema funzionale di un computer ad architettura data-flow dinamica.

te. Si può anche pensare che, a patto di avere disponibili sufficienti risorse, il parallelismo ottenuto sia il massimo ottenibile, in questo modo soddisferemo un ulteriore requisito, cioè la dipendenza delle prestazioni dell'algoritmo soltanto dalle risorse disponibili. L'eliminazione del concetto di «memorizzazione» delle variabili elimina in toto i problemi di condivisione della memoria e gli effetti collaterali propri delle architetture asincrone, che abbiamo già esaminato, per esempio, a proposito della coerenza dei dati nelle cache memory di un sistema multiprocessore.

L'informazione nell'ambiente data-flow è composta da «pacchetti operazioni» e «pacchetti dati». Un pacchetto operazione è formato dall'opcode dell'operazione, dal nome degli operandi e dalla destinazione del risultato. Un pacchetto dati è invece costituito dal valore del risultato e dalle sue destinazioni, per esempio il numero d'ordine di ciascuna istruzione come nei vecchi Basic tanto per intenderci. Per ottenere un alto parallelismo è necessario che molti di questi pacchetti siano scambiati contemporaneamente dalle varie risorse della macchina data-flow. Di conseguenza il computer data-flow avrà un'architettura a comunicazione di pacchetto che è uno dei tipi che abbiamo considerato nel discorso generale sui sistemi multiprocessore.

Architetture data-flow statica e dinamica

Esistono due maniere fondamentali per progettare un computer data-flow in relazione al modo in cui i pacchetti dati vengono utilizzati. In un tipo detto «data-flow statico», si assume che sugli archi che connettono un'istruzione alla successiva, non possa esistere più di un pacchetto dati per volta; questo implica che il flusso dei dati su quell'arco è bloccato finché tale istruzione non sarà eseguita, cioè quando tutti i dati non saranno presenti sugli archi di ingresso. Lo svantaggio maggiore è naturalmente nella esecuzione delle iterazioni dei cicli che non possono essere eseguite in parallelo, essendo costituite dalle stesse istruzioni. Questa inefficienza viene ripagata con una maggiore semplicità dei pacchetti dati che non debbono recare nessuna informazione temporale, visto che non esistono pacchetti in attesa con cui possano essere confusi. Tuttavia, dal punto di vista hardware, si deve prevedere un segnale di acknowledge tra le risorse per permetterne l'indispensabile sincronizzazione. L'altro tipo di architettura, ottenuto rimuovendo l'ipotesi di avere un solo dato per arco, è riferito come «data-flow dinamico». In tale situazione è necessario predisporre delle «etichette» che mantengano i dati ordinati, in modo che possano essere giustapposti per formare la corretta coppia di operandi da inviare all'unità funzionale che deve eseguire l'istruzione. In questo modo si riesce ad ottenere il massimo parallelismo possibile, ed inoltre si ottiene di svolgere in parallelo le iterazioni di un ciclo se la dipendenza dei dati lo consente, semplicemente etichettando progressivamente i dati con l'indice della iterazione a cui appartengono.

Queste due organizzazioni sono basate su due differenti schemi per sincronizzare l'esecuzione delle istruzioni. Un punto comune, del resto indispensabile per qualificare un elaboratore come parallelo, è riscontrabile nel fatto che sono presenti un insieme di processori in grado di computare asincronamente i pacchetti istruzione. In figura 4 troviamo lo schema funzionale di una macchina data-flow statica; i pacchetti dati sono contenuti nell'«unità di aggiornamento» che li distribuisce alle rispettive destinazioni nella «memoria istruzioni»; non appena un'istruzione ivi contenuta ha ricevuto tutti gli operandi richiesti, trasmette la sua disponibilità alla «unità di fetch» che si incarica di prelevarla dalla «memoria istruzioni» per inserirla nella «coda delle istruzioni abilitate». Tale istruzione è poi prelevata dal primo processore pronto nell'«unità di processo» che provvede a eseguire l'istruzione stessa e a passare il risultato, un pacchetto dati, all'«unità di aggiornamento».

Nell'organizzazione data-flow dinamica, di cui uno schema funzionale è esposto in figura 5, la sincronizzazione del sistema è basata su un meccanismo di verifica e accoppiamento, che chiameremo con il termine inglese «matching» per uniformità con la letteratura sull'argomento. L'«unità di matching» accetta in ingresso i pacchetti dati e controlla se ciascuno di questi può essere accoppiato con uno di quelli precedentemente memorizzati, altrimenti lo conserva per successivi confronti. La coppia di operandi formata dall'«unità di matching», viene inviata alla «unità di fetch e di aggiornamento» che preleva dall'«unità di memoria» l'istruzione corrispondente e accoda istruzione ed operandi nella «coda delle istruzioni abilitate». L'«unità di processing» preleverà da tale coda l'opcode e gli operandi

rispettivi per eseguire materialmente l'istruzione, il risultato di questa sarà poi inviato all'«unità di matching».

Come si vede entrambe le architetture hanno una struttura pipeline ad anello. Se includiamo una sezione di I/O otteniamo la struttura generalizzata di figura 6, in cui possiamo individuare quattro moduli principali: la memoria, i processori, la rete di interconnessione e l'unità di input/output. La memoria contiene i pacchetti istruzioni, i processori costituiscono l'unità di calcolo per l'escuzione in parallelo delle istruzioni pronte. La rete d'interconnessione è usata per trasmettere i risultati alle rispettive istruzioni destinazione e l'unità di input/output è l'interfaccia tra la macchina ed il mondo esterno. Nel modello dinamico questa unità provvede anche ad eseguire il matching dei pacchetti dati in quanto bisogna tener conto anche di eventuali input esterni.

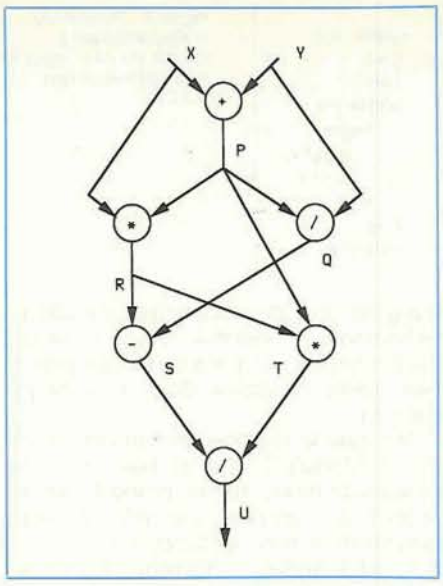


Figura 7 - Grafo data-flow per il calcolo di U= (X(X+Y)-(X+Y)/Y)/(X*(X+Y)*(X+Y)).*

Grafi e linguaggi data-flow

Per ottenere la massima efficienza da un'architettura data-flow è necessario che l'algoritmo venga espresso in maniera tale che il parallelismo implicito possa essere facilmente estratto. Da un punto di vista teorico, basterebbe conoscere il grafo della dipendenza dei dati di un programma perché una macchina data-flow possa computare il programma stesso. È certamente possibile ricavare tale grafo anche da un programma di tipo tradizionale, ma abbiamo già visto che sono necessari accorgimenti da parte di chi scrive il codice, per evitare situazioni inconsistenti o d'errore. Appare perciò utile la ricerca di un linguaggio che permetta di esprimere in maniera più diretta e naturale il parallelismo presente in un algoritmo. Diversi linguaggi sono stati proposti in letteratura quali l'Irvine Data-flow e il Value Algorithmic Language, tra quelli espressamente progettati per computer data-flow. Attualmente la tendenza è quella di implementare linguaggi funzionali più noti quali il Lisp, che negli ultimi anni ha raggiunto una certa diffusione.

Abbiamo già osservato che il massimo parallelismo si ottiene soltanto se la sequenza delle istruzioni è determinata dalla dipendenza dei dati fra di esse. Basandoci su questa affermazione, puntualizziamo un insieme di proprietà che i linguaggi data-flow devono possedere:
— libertà da effetti collaterali sui dati.
— Località degli effetti dei dati.
— Soddisfazione della regola della singola assegnazione.
— Esecuzione delle iterazioni dei cicli in parallelo.

I grafi data-flow

In un computer convenzionale, l'analisi del programma per ottimizzare il codice e l'utilizzo delle risorse, è quasi sempre effettuata a tempo di compilazione. Se eseguissimo tale analisi sul seguente frammento di programma:
1. p=x+y deve attendere gli input x e y
2. q=p/q deve attendere il completamento dell'istruzione 1
3. r=x*p deve attendere il completamento dell'istruzione 1
4. s=r−q deve attendere il completamento dell'istruzione 2 e 3
5. t=r*p deve attendere il completamento dell'istruzione 3
6. u=s/t deve attendere il completamento dell'istruzione 4 e 5

scopriremmo le elencate dipendenze dei dati e potremmo estrarre queste cinque diverse, ma equivalenti sequenze di esecuzione seriale delle istruzioni:

(1,2,3,4,5,6)
(1,3,2,5,4,6)
(1,3,5,2,4,6)
(1,2,3,5,4,6)
(1,3,2,4,5,6)

Su un computer parallelo è possibile eseguire le sei istruzioni, in tre passi invece di sei:

(1, [2 e 3 simultaneamente], [4 e 5 simultaneamente], 6)

Un «grafo diretto di flusso» può essere un altro modo di rappresentare tale analisi, in figura 7 potete vederne il disegno. Precisiamo che un «grafo diretto di flusso» è un grafo diretto i nodi del quale corrispondono ad operatori e gli archi sono puntatori per la trasmissione dei dati. Il grafo mostra assai chiaramente i vincoli di sequenziamento delle istruzioni. In una macchina data-flow tale grafo è il vero e proprio programma a basso livello e l'esecuzione delle istruzioni sui nodi è determinata dalla disponibilità dei dati.

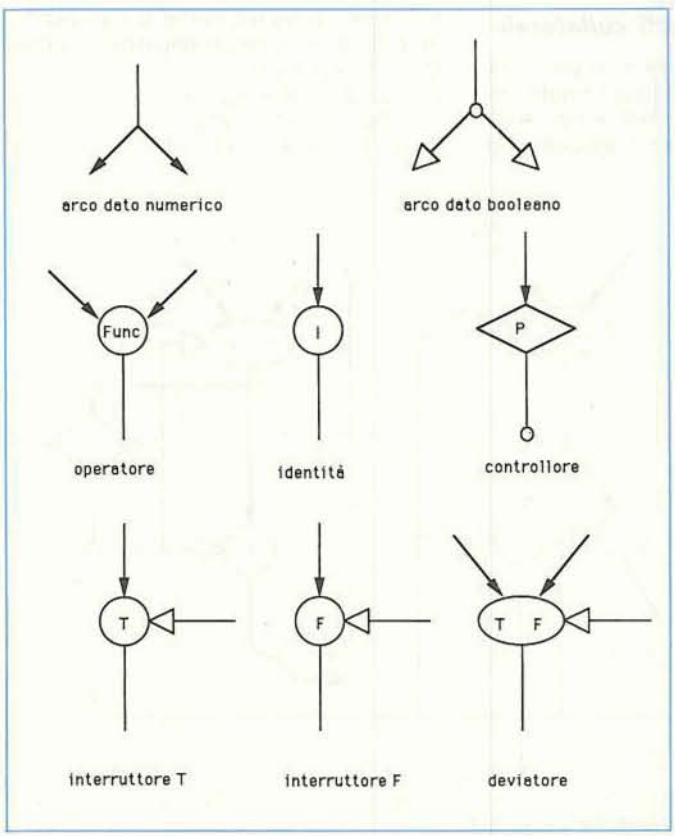


Figura 8 - Elementi costruttivi di un grafo data-flow.

Per eseguire un qualsiasi programma, è necessario un costrutto condizionale e di conseguenza il tipo booleano; a tal fine, nei grafi data-flow sono previsti due tipi di archi, per i dati numerici e per quelli booleani. In figura 8 vi sono vari tipi di operatori per costruire grafi data-flow. I più utili sono senza dubbio il «deviatore» e il «controllore» che permettono di costruire cicli condizionati. Il «deviatore» permette di connettere uno dei due input con l'uscita in relazione al valore dell'arco di controllo e il «controllore» restituisce un valore booleano, risultato del test eseguito applicando il predicato P all'ingresso. Gli «interruttori» T ed F lasciano passare il dato in ingresso soltanto se il valore presente sull'arco di controllo è rispettivamente True o False. In figura 9 e figura 10 potete trovare il listato e il rispettivo grafo data-flow di un programma per il calcolo dell'espressione z=x^n. Notate in particolare che i deviatori sugli archi di ingresso al programma permettono di immettere le variabili x ed n e di assegnare 1 alla y, semplicemente inizializzando i «pacchetti di controllo», cioè i dati che vanno sugli archi di controllo, pari a False.

Analizziamo ora meglio il significato dei requisiti di un'implementazione di linguaggio data-flow.

Località degli effetti e libertà degli effetti collaterali

La località delle variabili implica che l'effetto di una variabile sia limitato soltanto a contesti ben definiti e non abbia effetti collaterali. Molti linguaggi raggiungono questo scopo disponendo la definizione di variabili locali, fissando rigidamente la visibilità della variabile al frammento di codice dove è stata dichiarata.

```
input x,n
y:=1
i:=n
while i>0
   begin
      y:=y*x
      i:=i-1
   end
z:=y
output z
```

Figura 9 - Programma in Pascal-like per il calcolo di z=x^n. Input e output sono macro di I/O.

Per questa ragione un linguaggio come il Modula-2 che si basa su una modularità molto spinta, potrebbe essere un buon canditato per un'implementazione data-flow. Evitando quindi l'uso di variabili globali si dovrebbe controllare la possibilità di generare effetti collaterali; tuttavia un'altra potenziale sorgente di problemi può derivare dall'utilizzo di variabili passate a procedure per «riferimento» cioè tramite l'indirizzo di memoria invece che per «valore». In altre parole bisogna isolare totalmente l'input e l'output di ogni procedura.

Regola dell'assegnazione singola

Questa regola vieta l'utilizzo sul lato sinistro di una istruzione, dello stesso nome di variabile più di una volta. In altre parole, ogniqualvolta si incontra una ridefinizione di una variabile si deve scegliere un nuovo nome e riassegnarne tutte le occorrenze opportunamente. Ecco un esempio:

```
x:= p-q      q x:=p-q
x:=x*y    → x1:= x*y
w: x-y    → w:= x1-y
```

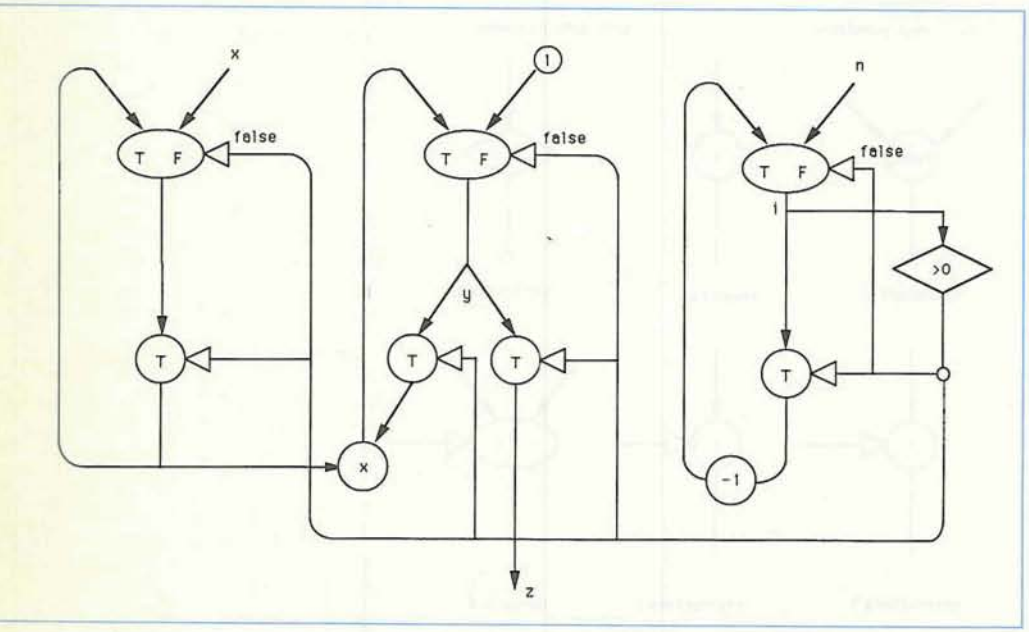


Figura 10 - Grafo data-flow del programma in figura 9.

La chiarezza e la possibilità di verificare facilmente la correttezza del programma controbilanciano notevolmente il fatto di non poter usare lo stesso nome per riassegnare una variabile. Sono stati comunque sviluppati tool automatici di assegnazione singola, basati su algoritmi proposti da Tesler ed Enea nel 1968.

Esecuzione parallela delle iterazioni dei cicli

I cicli costituiscono spesso il cuore degli algoritmi anzi gli algoritmi più pesanti, quelli che quindi si vorrebbe maggiormente parallelizzare, risultano costituiti da un insieme di istruzioni eseguito ciclicamente un numero di volte molto elevato. È necessario perciò trovare un modo tale che le iterazioni possano essere eseguite in parallelo, dipendenza dei dati permettendo. Considerando il fatto che le macchine data-flow dinamiche hanno pacchetti etichettati, una maniera semplice per conseguire questo scopo, consiste nel costruire la destinazione aggiungendo alle informazioni per localizzare l'istruzione, anche un'informazione sul contesto di appartenenza dell'istruzione:
per esempio se si esegue un'iterazione controllata da indice, l'indice stesso. Si ottiene in tal modo uno «spiegamento» del ciclo in frammenti di codice replicati tante volte quante sono le iterazioni previste.

Conclusioni

Il concetto di elaborazione «data-driven» costituisce probabilmente il modo più nuovo e rivoluzionario di considerare la computazione. Anche se l'idea alla base è molto semplice ed efficace, e le architetture eleganti, tuttavia le macchine «data-flow» difficilmente potranno uscire dai laboratori di ricerca per due motivi fondamentali, uno progettuale ed un altro per così dire sociale. I programmi data-flow tendono ad occupare parecchia memoria a causa della duplicazione dai dati passati per valore e dello «spiegamento» dei cicli; inoltre, l'idiosincrasia del «programmatore medio» ad apprendere linguaggi funzionali e nuovi modi di programmare, non favoriscono certo la diffusione su larga scala di queste architetture innovative. Le ricerche correnti sono allora rivolte alla realizzazione di strumenti per l'estrazione dei grafi data-flow dai programmi scritti nei linguaggi più convenzionali come il Fortran, per la messa a punto di linguaggi intuitivi e potenti e di architetture che sfruttino l'efficacia del data-flow più a livello di processi e procedure che non di singole istruzioni, semplificando tra l'altro la progettazione di hardware appositamente dedicato.

NOTAI

ICP

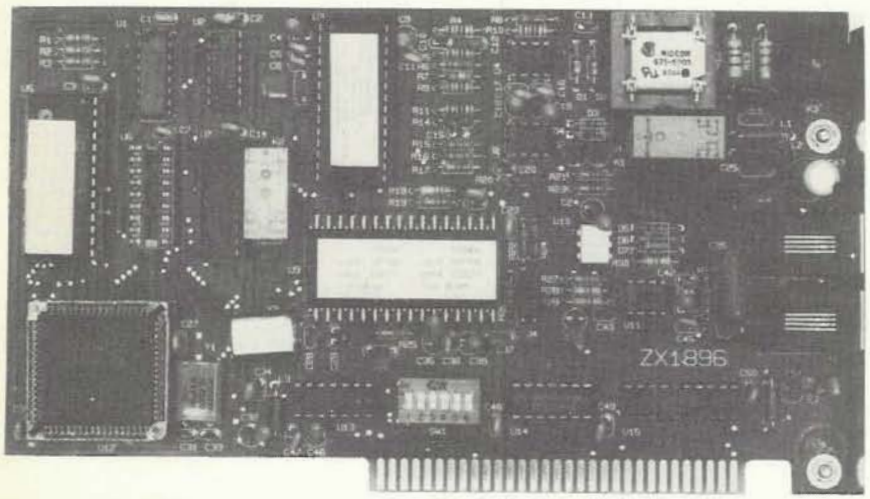
international computer products s.r.l.
Via Dei Berio, 97 - 00155 Roma
Tel. 06/2252291 - 10 linee r.a.
Via Ecetra, 24/26 - 04100 Latina
Tel. 0773/486977 - 487510

SISTEMA PER L'AUTOMAZIONE DEGLI STUDI NOTARILI

VI ATTENDIAMO ALLO
SMAU
(MILANO, 4/8 OTTOBRE)
PADIGLIONE 17 - STAND D25
CONSORZIO SOFTWARE

VERSIONE 4.0

Con questa scheda il tuo PC riceve e trasmette i fax!



- Riceve e stampa automaticamente i fax in arrivo, poi li salva su disco!
- Ruota il fax di 90 o 180 gradi per poter vedere sullo schermo il fax in arrivo anche se è stato trasmesso sottosopra o orizzontalmente!
- Con il programma Bit Paint (optional), consente di vedere e ricevere immagini, modificarle, commentarle e poi ritrasmetterle al mittente!
- Fax Mail Merge consente la spedizione di fax personalizzati a più indirizzi!

...ed è anche un modem a 2400 bps!

- Velocità di trasmissione fax 9600 bps!
- Maggior risoluzione di un normale apparecchio telefax
- Trasmissione in differita per trasmettere nelle fasce orarie di minor costo!
- Durante la ricezione di un fax consente di utilizzare altri programmi!
- Trasmette ad apparecchi fax G3 ognuno di questi tipi di files: ASCII (testo), PCX, IMG, TIFF e FAX. I files PCX a colori sono automaticamente convertiti in scala di grigi!

IN OFFERTA DI LANCIO 750.000 LIRE INCLUSO SOFTWARE BIT-FAX

IVA ESCLUSA

Puoi acquistare il faxmodem agli indirizzi indicati o per maggiori informazioni rivolgiti a:

Microforum
MANUFACTURING INC.
TORONTO - CANADA

1 Woodborough Avenue, Toronto, Canada M6M 5A1
Tel. 001 416 656 6406 Fax 001 416 656 6368 Telex 06 23303

Media Disk Antonelli
12, Via Ciociaria - 00162 Roma
Telefono 06/4240379

Floppy's Market
5, P.za del Popolo
56029 S.Croce sull'Arno (PI)
Tel. 0571/35124 Fax 32768

Non Stop spa
11,Via B. Buozzi
40057 Cadriano di Granarolo (BO)
Tel. 051/765299 Fax 765252

SPY Cash & Carry
Piazza Arenella 6/A
80055 Portici (NA)
Tel. 081/5785623 Fax 5785167