Le architetture pipeline

di Giuseppe Cardinale Ciccotti

Fra le varie possibili architetture dei sistemi di calcolo paralleli possiamo riconoscere sicuramente tre classi di base:
Pipeline Computers
Array Processors
Multiprocessors Computers.
Queste categorie sono sufficientemente concettuali in modo da consentirci di valutare in maniera virtualmente indipendente dalle singole macchine quali implicazioni comporti progettare, programmare e usare tali computer. A questo scopo analizzeremo in dettaglio, in questo e nei successivi appuntamenti, le suddette classi

Vettorizzazione

I Pipeline Computers sfruttano un parallelismo temporale eseguendo più operazioni «sovrapposte» negli stessi intervalli di tempo. Come è stato già espresso nel numero precedente e in diverse altre occasioni in questa stessa rubrica, una pipeline è assimilabile ad una catena di montaggio. Faremo tuttavia un esempio per fugare ogni dubbio residuo e per confrontarla con un'architettura von Neumann, tipicamente non pipeline. Supponiamo perciò di avere disponibili 4 unità di calcolo, chiamati PE (Processor Elements), disposte in pipeline come in figura 1. Vogliamo inoltre eseguire un semplice programma come questo:

```
for (i=0; i<1000)
{
 r1[i]=a*b;
 r2[i]=r1[i]+c;
 r3[i]=r2[i]/d;
 risultato[i]=r3[i]-f
}
```

Dove l'array è inizializzato a 0 e a,b,c,d ed f sono costanti. Come si vede facilmente il numero di operazioni da eseguire è 1000·4=4000 (escludendo l'incremento e il controllo della variabile i). Poniamo, per semplicità, che tutte le operazioni richiedano un tempo costante, t_i. Un solo PE che deve completare un'istruzione prima di eseguire la successiva, porterebbe a termine il nostro programma in 4000·t_i, calcolando un elemento del vettore risultato ogni 4·t_i. Nella struttura a pipeline, invece, istruzioni successive sono sovrapposte nel tempo, come è evidenziato in figura 2, dove potete vedere la differenza tra il flusso delle istruzioni in un sistema pipeline e in un sistema non pipeline. Il parallelismo a cui ci riferiamo in questo esempio, è di tipo intraistruzione: un processo, il ciclo che vogliamo eseguire, è smembrato in vari task, le istruzioni del ciclo, eseguite dalla pipeline con un parallelismo temporale. Si noti come il primo elemento del vettore risultato è comunque disponibile (nel nostro esempio) dopo 4·t_i, questo tempo è detto tempo di latenza, mentre i successivi vengono forniti ogni t_i, l'ultimo uscirà dopo 999·t_i; il tempo complessivo di esecuzione dell'algoritmo è perciò di 4·t_i+999·t_i=1003·t_i. Questi tempi mettono in evidenza una caratteristica negativa nell'architettura pipeline: il tempo di latenza cioè il tempo necessario affinché la pipeline vada a regime o, come si usa dire si «instauri» la pipeline, impedisce che lo speed-up raggiunga il massimo teorico. In questo caso otteniamo uno speed-up pari a 4000·t_i/1003·t_i=3.988 inferiore seppur di poco a 4 che costituisce il valore massimo raggiungibile con 4 PE. Si può facilmente verificare, basta cambiare nel programma precedente il numero di iterazione e rifare i conti, come lo speed-up si approssimi al valore massimo, quanto più a lungo si mantiene la pipeline a regime rispetto al tempo di latenza. In figura 3 potete vedere come varia, nel nostro esempio, lo speed-up in funzione del numero di iterazioni. Il procedimento appena descritto viene chiamato vettorizzazione dei loop, i calcolatori che sfruttano tale parallelismo sono detti Vector Processor. È una tecnica molto usata su cui è stata basata la generazione di supercomputer precedente alla attuale. Come abbiamo visto è semplice ed efficace, inoltre molti degli algoritmi di tipo scientifico prevedono lunghi cicli di calcolo. Queste caratteristiche hanno spinto a produrre dei linguaggi e dei compilatori che facilitassero ed eseguissero automaticamente la procedura di vettorizzazione, come ad esempio il VECTRAN (VECtorizing ForTRAN).

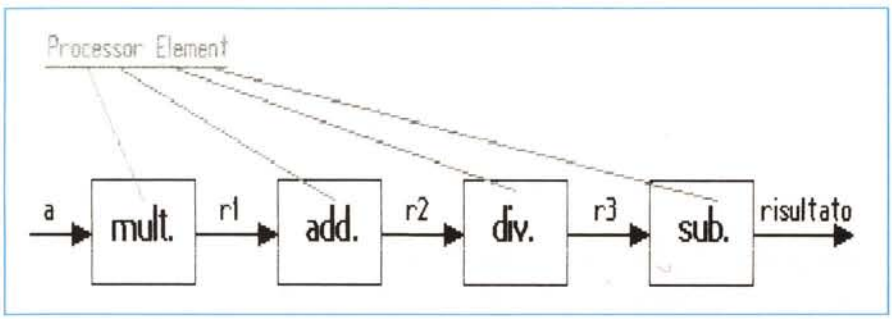


Figura 1 - Pipeline 4 stadi. I PE eseguono 4 diverse operazioni negli stessi intervalli di tempo.

Progetto di una pipeline

Tuttavia come i più sagaci lettori avranno già istituito, l'architettura pipeline presenta un grave handicap, che ne limita l'uso in maniera determinante: è completamente inadatta ad eseguire programmi che prevedano molte istruzioni di salto condizionato. L'effetto di tali istruzioni è di distruggere la pipeline, infatti ciò provoca l'esecuzione di un

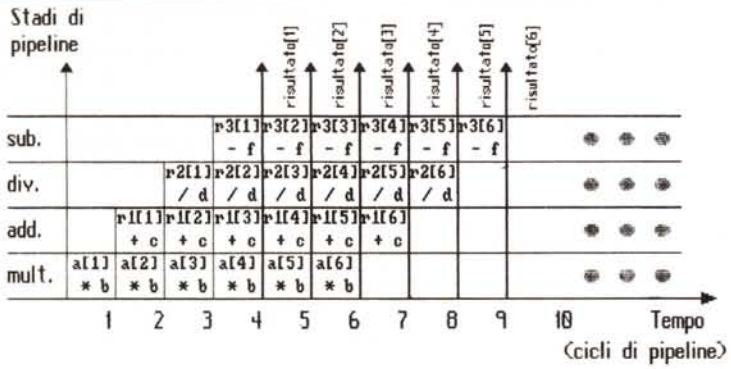


Diagramma spazio-tempo per un pipeline computer

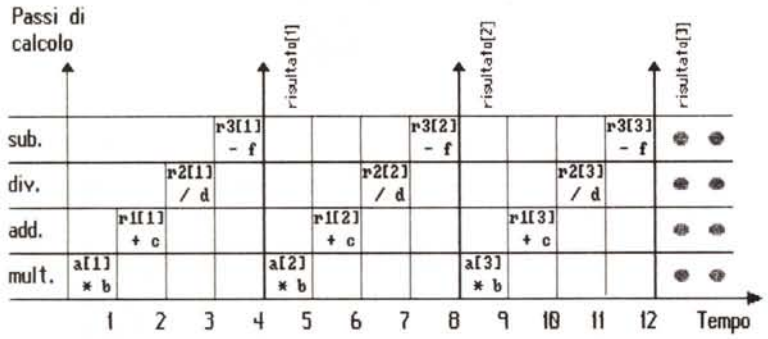


Diagramma spazio-tempo per un computer Von-Neumann

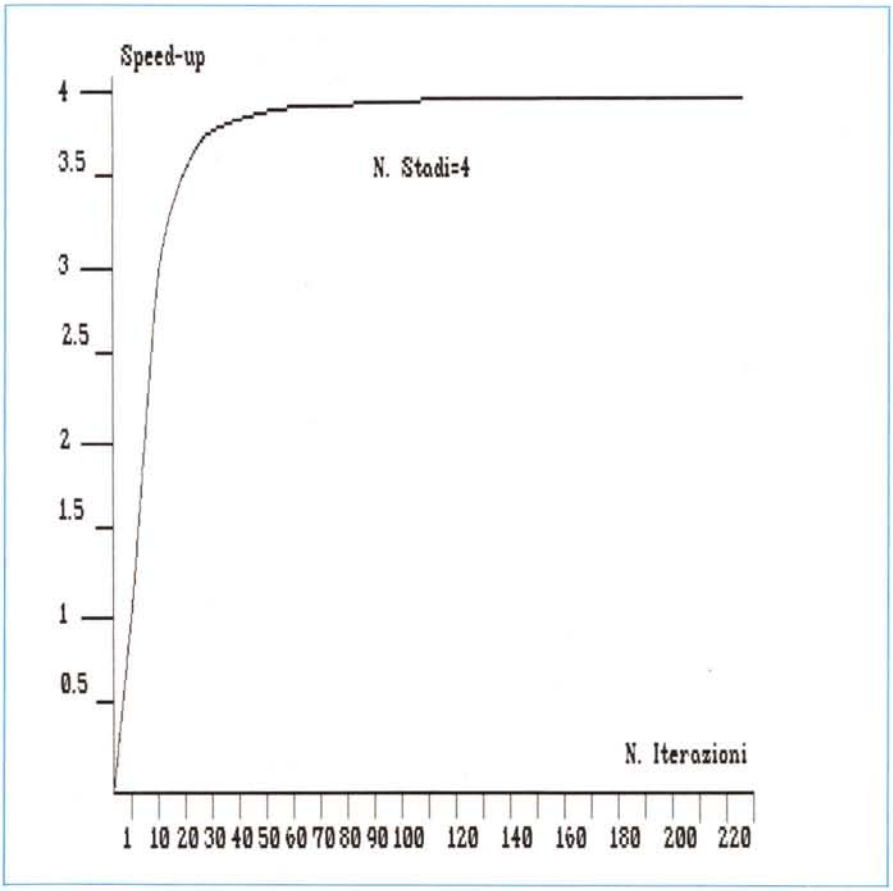
Figura 2 - Esecuzione del programma d'esempio in una architettura pipeline e una von Neumann. Notate la riduzione del tempo di calcolo.

Figura 3 - Curva di speed-up al variare delle iterazioni in pipeline.

differente segmento di programma non prevedibile a priori con conseguente tempo di latenza dovuto all'instaurazione di una nuova pipeline.

Se i salti sono frequenti si perde quasi del tutto il vantaggio di avere più processori, addirittura la somma dei tempi di latenza può diventare preponderante rispetto al tempo di calcolo effettivo così da approssimare lo speed-up all'unità: in tal caso significa che un solo PE è più efficiente della pipeline nell'esecuzione di quel programma. Questo è il principale motivo che rende inadeguati i Pipeline Computers come general purpose computer. Tuttavia la architettura pipeline è lo schema per eccellenza delle CPU ad alte prestazioni. I processori della famiglia 68000, i nuovi Intel 80680 e Motorola 88000, l'AMD 29000, per non parlare dei DSP (Data Signal Processing) o dei processori grafici come il NEC 72120, fanno uso di pipeline per sovrapporre le fasi di fetch ed execute di ogni istruzione. Con tale accorgimento, detto prefetch, le istruzioni di salto hanno un effetto tanto più controproducente quanto più è lunga la pipeline. Per questo motivo i più potenti microprocessori come l'AMD 29000 prevedono, in corrispondenza di istruzioni di salto condizionato, il prefecth delle istruzioni delle diverse diramazioni.

Nell'esempio che abbiamo considerato, abbiamo supposto che tutti i PE impiegassero lo stesso tempo nell'esecuzione delle varie istruzioni. Rilasciamo ora questa ipotesi, per nulla realistica, e valutiamone le implicazioni. Appare subito chiaro che tutta la pipeline dovrà essere sincronizzata con il PE che impiega più tempo ad eseguire i suoi compiti. Se così non fosse tale PE non riuscirebbe a produrre i dati con la stessa frequenza con la quale li riceve e il flusso nella pipeline si interromperebbe. Se in una realizzazione pratica implementiamo i generici PE che abbiamo finora considerato, con dei microprocessori commerciali, per esempio dei 68000, ci troviamo a fissare due frequenze di clock di un sistema così definito. Non è un errore di stampa, in effetti dobbiamo tenere conto di due temporizzazioni: una propria dei singoli microprocessori presenti in ciascun stadio della pipeline, che perciò chiameremo stage clock, e un'altra, detta pipe



clock, che scandisce la trasmissione dei dati da uno stadio al successivo nella pipeline. Lo stage clock è grosso modo fissato dal costruttore del componente che stiamo usando: se per esempio adoperiamo dei 68000 a 16 MHz, sceglieremo lo stage clock poco al di sotto di tale frequenza, per sfruttare al massimo le capacità elaborative del microprocessore. La scelta invece del pipe clock è responsabilità del progettista hardware e dipende da quello che si vuole ottenere. Se la nostra pipeline costituisce il «cuore» di un general purpose computer, non sappiamo in generale quali cicli di istruzioni verranno vettorizzati; dobbiamo perciò cautelarci mettendoci nel caso peggiore. Tale situazione si verifica quando due stadi contigui eseguono uno l'istruzione più breve e il successivo quella più lunga di tutto il set. Il pipe clock sara perciò uguale al tempo di esecuzione dell'istruzione più

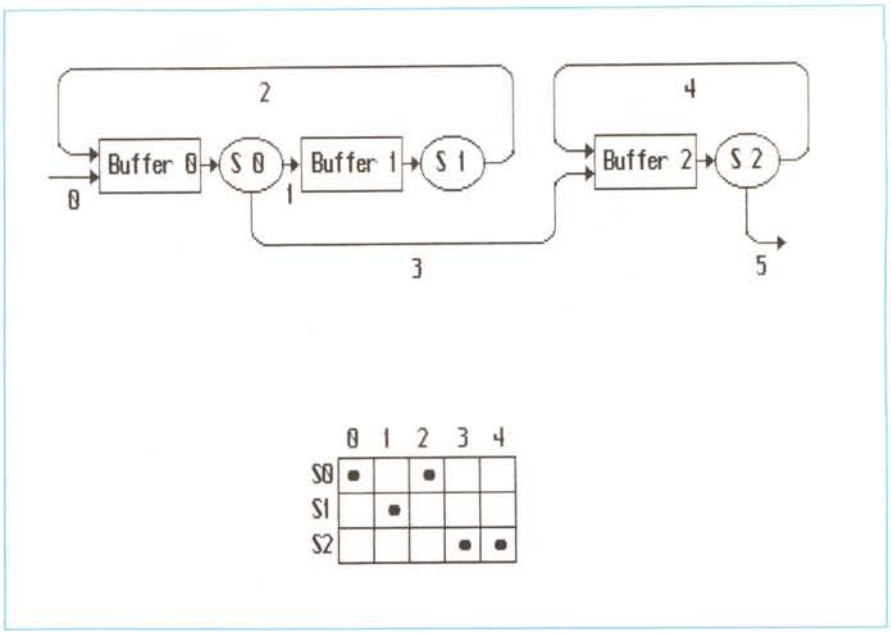


Figura 5 - Pipeline multifunzionale e relativa tabella di scheduling. L'ordine dei vari stadi può essere narrangiato per mezzo di connessioni programmabili. La tabella mostra quali task sono allocati negli stadi della pipeline per ogni slot temporale.

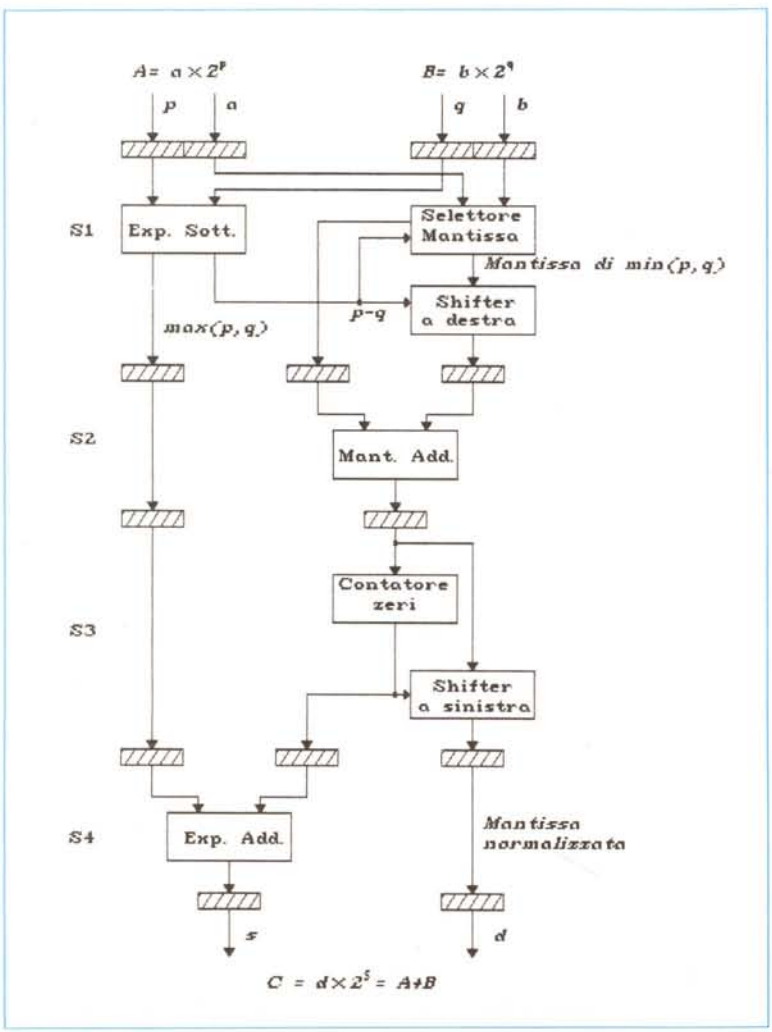


Figura 4 - Addizionatore floating point con pipeline 4 stadi specializzati.

lunga del repertorio del microprocessore, in tal modo siamo sicuri che la pipeline non subirà mai dei guasti dovuti a mancanza di sincronizzazione fra i vari stadi: ogni stadio attenderà un certo numero di cicli, pari alla differenza tra il numero di cicli dell'istruzione più lunga e il numero di cicli dell'istruzione che è stata appena eseguita, prima di trasferire il risultato della propria istruzione allo stadio successivo. In questa maniera però ogni istruzione, anche la più semplice, diventerebbe lenta quanto la più complessa con ovvio scadimento delle prestazioni complessive. In tal caso i vantaggi derivanti dall'uso di una pipeline andrebbero verificati caso per caso.

Useremo quindi una formula più generale per valutare lo speed-up:

$$\text{speed-up} = \frac{T_1}{T_k} = \frac{t_0+t_1+\ldots+t_n}{[k+(n-1)]\cdot t_{max}}$$

dove t_i (i=0..n) indica il tempo di esecuzione di ciascuna delle n+1 istruzioni del programma seriale mentre tmax è il tempo fisso di esecuzione delle istruzioni in pipeline vale a dire il pipe clock. Se lo speed-up è maggiore di uno allora converrà utilizzare una pipeline per eseguire il programma. Tuttavia come le

Bibliografia

Hwang K., Briggs F. «Computer Architecture and Parallel Processing», McGraw-Hill, 1988
Ewinger W., Haan O., Haupenthal E. and Siemers C., «Modelling and Measurement of Memory Access in SIEMENS VP Supercomputers», Parallel Computing, vol. 11 n. 3 1989, pp. 361-365.

statistiche effettuate su diverse applicazioni e l'esperienza di programmazione in Assembler ci insegnano, l'istruzione più complicata è in generale quella meno eseguita del set. Ci troviamo quindi di fronte ad un controsenso: vogliamo incrementare la velocità di esecuzione di un programma e per effettuare una qualsiasi istruzione semplice, per esempio un CMP (compare), impieghiamo lo stesso tempo di una MULS (moltiplicazione con segno)!

Pipeline unifunzionale e multifunzionale

Per ovviare a questo problema ci sono due possibilità: pipeline unifunzionale oppure multifunzionale. Queste due tipologie rispecchiano due concetti antitetici e fondamentali della architettura dei calcolatori: unità semplici e veloci da un lato o unità potenti, ma più lente dall'altro. È la stessa disputa, che divide programmatori e progettisti, riguardo l'approccio RISC e CISC. Una pipeline unifunzionale esegue un solo task per volta, è caratterizzata da stadi limitatamente programmabili o addirittura specializzati nell'esecuzione efficiente di un solo tipo di operazione, collegando in cascata più stadi che effettuano operazioni diverse otteniamo un dispositivo adatto ad una certa operazione complessa. Tale architettura risulta semplice ed efficiente, i vari stadi specializzati saranno scelti con un tempo di esecuzione simile, perciò la logica di controllo sarà semplice e la computazione efficiente. Il prezzo che paghiamo però è una perdita di flessibilità infatti tale dispositivo non potrà eseguire compiti diversi da quelli per cui è stato progettato, in particolare la lunghezza della pipeline è fissa perciò non potremo effettuare compiti che usino solo alcuni stadi a meno di non rendere «trasparenti» quelli non usati, programmando delle istruzioni non operative; l'efficienza che ne consegue è molto bassa. Tale schema è molto usato per i dispositivi che eseguono operazioni in virgola mobile, in figura 4 potete vedere la pipeline di un addizionatore in floating point. Se quindi progettassimo un computer con pipeline unifunzionali, necessariamente ne dovremmo predisporre più d'una, implementando meccanismi di sincronismo e di collegamento fra di esse. Il famoso Cray-1 è basato su un'architettura di questo tipo, il meccanismo di gestione delle varie pipeline è detto chaining: le diverse pipeline specializzate vengono collegate opportunamente attraverso buffer intermedi ad alta velocità di accesso in cui vengono depositati i risultati intermedi fra l'uscita di una pipeline e l'ingresso della successiva.

Una pipeline multifunzionale, invece, presenta degli stadi non specializzati che possono essere completamente riprogrammati; inoltre, per aumentare la flessibilità, si prevedono in genere dei multiplexer fra uno stadio e il successivo in maniera tale che la lunghezza della pipeline possa essere modificata a piacere. La figura 5 mostra uno schema di questo tipo; la logica di controllo è molto più complicata rispetto a quella della pipeline unifunzionale. La pipeline multifunzionale è molto adatta al multitasking, la sua riconfigurabilità permette di strutturare il flusso dei dati nel modo più opportuno. I vari stadi sono allocati ai diversi task, collegando e arrangiando in qualunque modo gli stadi stessi. Così per esempio in figura 6, il processo A usa il primo e il secondo stadio mentre B tutti e tre. In tale maniera l'efficienza sarà mantenuta sempre alta poiché è possibile far lavorare tutta la pipeline, al contrario di quella unifunzionale; tuttavia la dipendenza fra i dati, la collisione nell'accesso a questi e soprattutto la concorrenza nell'allocazione degli stadi ai task impediscono che l'efficienza sia massima. L'argoritmo di scheduling eseguito dall'unità di controllo, deve necessariamente tenere conto di quali stadi devono essere allocati ai vari task per non interrompere il flusso della pipeline (o meglio di tutte le pipeline) e decidere il momento, detto slot temporale, in cui allocarli inserendo opportuni stadi di attesa possibilmente all'inizio dell'esecuzione del task. Esistono dei metodi di descrizione, attraverso dei diagrammi temporali, di questo meccanismo di scheduling, ma non risulta che siano stati proposti dispositivi commerciali che sfruttino, questa politica di allocazione delle risorse.

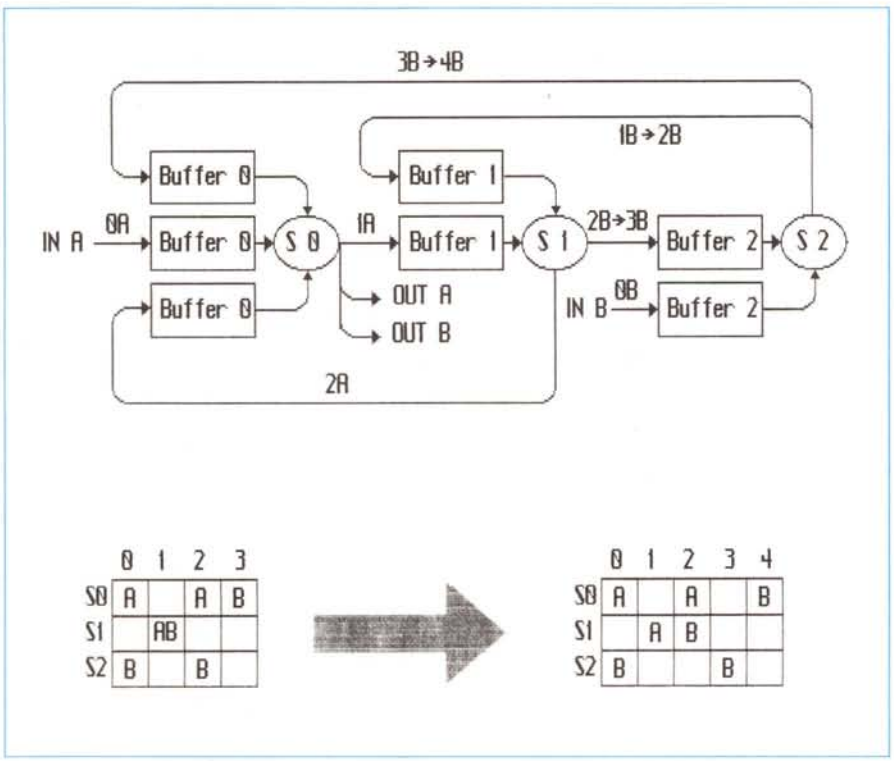


Figura 6 - Pipeline multifunzionale multitasking. Sono predisposti tanti buffer quanti sono necessari per evitare collisioni fra task stessi. La tabella di scheduling viene espansa quando più task richiedono lo stesso stadio nello stesso intervallo di tempo. I task collidenti sono allora allocati in differenti slot temporali.

Conclusioni

Abbiamo così descritto quali sono le principali caratteristiche dell'architettura pipeline. In particolare ci siamo resi conto di come le realizzazioni pratiche e le esigenze di flessibilità abbiano modificato il semplice concetto di «catena di montaggio» da cui siamo partiti. La complessità delle pipeline multifunzionali in effetti ci porta a considerare architetture diverse, ma ugualmente efficienti e flessibili che analizzeremo nelle puntate successive.