Architettura degli Elaboratori

Note su elaborazione in parallelo a livello firmware

con applicazioni al sottosistema di memoria

Queste note contengono integrazioni, precisazioni ed esempi riguardanti il materiale didattico del Capitolo X del libro di testo (fondamenti di elaborazione in parallelo) e sue applicazioni. In particolare:

- viene precisato come utilizzare, agli effetti di questo corso, il Capitolo X che di per sé ha obiettivi più ampi;
- vengono riportati esempi di utilizzo delle tecniche di parallelizzazione nella strutturazione firmware;
- vengono applicati i concetti di parallelismo allo studio del sottosistema di memoria e specificamente delle gerarchie di memoria con cache (Capitolo VIII).

Sommario

1 Sistemi operanti su stream e loro valutazione 2
1.1 Tempo di servizio ideale e tempo di servizio effettivo 2
1.1.1 Esempio: struttura pipeline 2
1.1.2 Funzionamento a regime e funzionamento transitorio 4
1.1.3 Impatto delle comunicazioni 5
1.1.4 Sistemi complessi e sottosistemi paralleli 5
1.2 Un esempio di sistema come grafo aciclico di unità 6
1.2.1 Risoluzione del grafo 7
1.2.2 Valutazione in presenza di colli di bottiglia 9
1.2.3 Parallelizzazione di colli di bottiglia: soluzione farm 9
1.2.4 Parallelismo data-flow 12
2 Applicazione al sottosistema di memoria 13
2.1 Studio delle prestazioni di una memoria interallacciata 13
2.2 Scritture in memoria sincrone e asincrone 14
2.2.1 STORE asincrona 14
2.2.2 Impatto tecnologico della STORE asincrona nei processori esistenti 15
3 Modello cliente-servente a domanda-risposta 16
4 Applicazione alla gerarchia di memoria con cache 17
4.1 Trasferimento di blocchi 17
4.2 Write-Through 18
4.3 Interconnessione memoria – CPU per il trasferimento di blocchi 19
4.4 Cache e Memory Mapped I/O 21

1 Sistemi operanti su stream e loro valutazione

In tutta la seconda parte del corso (gerarchie di memoria, parallelismo a livello di istruzioni) vengono utilizzati concetti fondamentali di strutturazione in parallelo e di valutazione delle prestazioni di sistemi operanti *su stream* (Cap. X, sez. 2, 3, 4).

Di particolare importanza è acquisire familiarità con i concetti di banda (tempo di servizio), latenza e loro valutazione tanto in sistemi strutturati a grafo aciclico quanto in sistemi cliente-servente.

1.1 Tempo di servizio ideale e tempo di servizio effettivo

D'ora in poi, quello che nella prima parte del corso è stato chiamato tempo medio di elaborazione deve essere valutato come **tempo di servizio**, distinguendolo dalla **latenza** quando il sottosistema studiato abbia un grado di parallelismo maggiore di uno.

Più precisamente, consideriamo il procedimento con il quale viene progettata una singola unità di elaborazione U (modulo di elaborazione a livello firmware): il tempo medio di elaborazione di U è il suo **tempo di servizio ideale**, $T_{s\text{-}id}$, con questo intendendo il tempo di servizio senza considerare l'appartenenza di U ad un sistema di unità cooperanti. Il **tempo di servizio effettivo** T_s di U tiene conto del fatto che U appartiene ad un sistema complesso, e coincide con il suo *tempo di interpartenza*, valutato come:

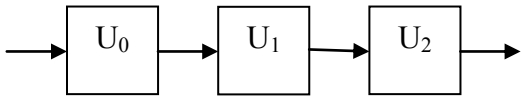
$$T_s = T_p = max\ (T_{s\text{-}id},\ T_A)$$

dove T_A è il *tempo di interarrivo* a U.

In altri termini: la *banda ideale* di U, uguale a $1/T_{s\text{-}id}$, esprime il numero massimo di operazioni esterne che U è in grado di elaborare nell'unità di tempo, cioè la sua capacità massima di elaborazione o **banda offerta**. Se la **banda richiesta**, cioè la frequenza $\lambda = 1/T_A$ di richieste di operazioni esterne, è minore o uguale alla banda offerta, allora di fatto U non può produrre più di $1/T_A$ risultati nell'unità di tempo; se invece la banda richiesta è maggiore della banda offerta, U non può produrre più di $1/T_{s\text{-}id}$ risultati nell'unità di tempo.

1.1.1 Esempio: struttura pipeline

Un semplice esempio di sistema Σ operante su stream (Cap. X, sez. 2) è la seguente struttura *pipeline*:



Gli *stadi* del pipeline sono realizzati da tre unità operanti in parallelo, ognuna specializzata nell'elaborazione di una certa funzione (F_0, F_1, F_2), che può anche avere uno stato.

Per ogni valore x dello stream di ingresso, il corrispondente valore dello stream di uscita è dato da $y = F_2(F_1(F_0(x)))$.

Nello stesso istante le tre unità sono impegnate nell'elaborazione delle rispettive funzioni su tre valori x_i, x_{i+1}, x_{i+2} distinti dello stream d'ingresso.

Per una struttura del genere ci interessa valutare le prestazioni a partire da informazioni sulle prestazioni delle singole unità (informazioni che sono note dalla loro progettazione separata), oltre che da altri parametri legati alla frequenza con cui arrivano i valori in ingesso e alle latenze di trasmissione.

Supponiamo che l'unità U_0 valuti il numero di occorrenze del valore *x* dello stream di ingresso in un array *A[N]* realizzato come componente logico memoria al suo interno. Sappiamo che (senza per ora considerare la latenza delle comunicazioni) il suo tempo di servizio ideale vale

$$T_{0-id} \sim N\,\tau$$

Noto il tempo di interarrivo T_A a Σ, una prima valutazione del tempo di servizio effettivo di U_0 può essere la seguente: se $T_A \geq N\ \tau$, allora il tempo di servizio effettivo T_0 vale T_A, altrimenti T_0 vale $N\ \tau$.

Questa analisi, però, non è necessariamente corretta, in quanto non tiene ancora del tutto conto dell'appartenenza di U_0 al sistema complessivo Σ.

Siano noti i tempi di servizio ideali di U_1 e U_2: T_1-*id* e T_2-*id*.

Il tempo di servizio effettivo di U_0, prima determinato, è per definizione il suo tempo di interpartenza, quindi è il *tempo di interarrivo* a U_1. Se:

$$T_0 \ \geq\ T_{1-id}$$

allora il tempo di servizio ideale di U_1 vale:

$$T_1 = T_0$$

A sua volta, questo è il valore del tempo di interarrivo a U_2. Se:

$$T_1 = T_0 \ \geq\ T_{2-id}$$

allora il tempo di servizio ideale di U_2 vale:

$$T_2 = T_1 = T_0$$

che quindi, essendo il valore del tempo di interpartenza dall'intero sistema, è il tempo di servizio di Σ. La banda di Σ vale dunque $1/T_0$.

Le ipotesi fatte circa le relazioni sui tempi di servizio sono valide solo nel caso in cui le unità U_1 e U_2 non provochino un abbassamento di banda rispetto alla banda di U_0. In tali ipotesi, la valutazione fatta del tempo di servizio di U_0 e dell'intero Σ è corretta.

Supponiamo invece che

$$T_0 < T_{1-id}$$

Il tempo di interpartenza di U_1 vale ora $T_{1\text{-}id}$, e questo è anche il tempo di interarrivo a U_2. Supponendo che $T_{2\text{-}id} < T_{1\text{-}id}$, il tempo di interpartenza da U2, quindi il tempo di servizio di Σ, vale $T_{1\text{-}id}$. Non solo: bisogna anche considerare che, trascorso un periodo <u>*transitorio*</u> di assestamento, anche *la banda di* U_0 *deve adeguarsi alla banda di* U_1, in quanto U_0, ogni volta che vuole inviare un valore a U_1, rimane mediamente bloccato che U_1 abbia completato l'elaborazione in corso e sia disponibile a ricevere un nuovo messaggio. In conclusione, <u>*a regime*</u> tutte le unità sono caratterizzate dalla stessa banda e questa

coincide con la minore banda tra le bande ideali delle singole unità e la frequenza di interarrivo. Cioè, *in generale* vale la seguente relazione:

$$T_\Sigma = \max(T_A, T_{0-id}, T_{1-id}, T_{2-id})$$

In termini equivalenti, l'analisi del sistema può essere fatta nel seguente modo: la *banda richiesta* a Σ vale

$$B_{richiesta} = \frac{1}{T_A}$$

La *banda offerta* da Σ vale

$$B_{offerta} = \frac{1}{\max(T_{0-id}, T_{1-id}, T_{2-id})}$$

La banda effettiva della computazione è la minore tra la banda richiesta e la banda offerta:

$$B_\Sigma = \min(B_{richiesta}, B_{offerta})$$

che equivale all'espressione precedente per T_Σ.

1.1.2 Funzionamento a regime e funzionamento transitorio

I concetti discussi ed esemplificati precedentemente sono in relazione con lo studio di una unità U (in generale un sottosistema) vista come *servente in un sistema a coda.*

Fondamentale è ragionare in termini di fattore di utilizzazione del servente, definito come:

$$\rho = \frac{T_{s-id}}{T_A}$$

Se $\rho \leq 1$, U non provoca alcuna diminuzione di banda del sistema costituito da U e dal sottosistema che genera lo stream (gli stream) d'ingresso a U, quindi il tempo di servizio *effettivo* vale

$$T_s = T_A$$

In questo caso, l'efficienza relativa di U vale proprio ρ, come si dimostra facilmente partendo dalla definizione:

$$\varepsilon_U = \frac{T_{s-id}}{T_s} = \frac{T_{s-id}}{T_A} = \rho$$

Se $\rho > 1$, U è un *collo di bottiglia* del sistema suddetto (provoca una diminuzione di banda del sistema rispetto al valore ideale), quindi

$$T_s = T_{s-id}$$

$$\varepsilon_U = 1$$

È importante distinguere il *funzionamento a regime* rispetto al *funzionamento nel transitorio.*

Se U è collo di bottiglia, inizialmente il mittente U_{source} dello stream non risentirà apprezzabilmente di questa situazione ma, a regime, U_{source} dovrà forzatamente adeguarsi

alla banda di U (in termini strutturali, U_{source} è mediamente bloccato in attesa dell'ack ogni volta che tenta di inviare un nuovo valore a U), e quindi *a regime* il tempo di interpartenza da U_{source} verso U, e quindi il tempo di interarrivo a U, diventerà uguale a $T_{s\text{-}id}$.

Ciò significa che, *se $\rho > 1$, l'analisi dei colli di bottiglia mediante la valutazione di ρ vale solo nel transitorio.*

Se invece U non è collo di bottiglia, il funzionamento a regime sarà statisticamente uguale a quello nel transitorio.

1.1.3 Impatto delle comunicazioni

Nel Cap. X, sez. 13, viene valutata la latenza di comunicazione tra due unità aventi lo stesso ciclo di clock τ:

$$L_{com} = 2\,(\tau + T_{tr})$$

con T_{tr} latenza di trasmissione del collegamento. Questo parametro esprime l'intervallo di tempo minimo tra l'invio di due messaggi consecutivi: dopo aver inviato un messaggio, il mittente non può inviare un nuovo messaggio prima che sia mediamente trascorso L_{com}. Ciò significa che, tenendo conto delle comunicazioni, il tempo di servizio ideale di una unità viene valutato come:

$$T_{s-id} = \max(T_{s-id-0}, L_{com})$$

dove $T_{s\text{-}id\text{-}0}$ è il tempo di servizio ideale senza tenere conto dell'impatto delle comunicazioni (quello che, nella prima parte del corso, veniva valutato come tempo medio di elaborazione).

È importante osservare che la valutazione di L_{com} ha validità generale nel funzionamento *a regime* (che esistano o meno colli di bottiglia).

Nell'esempio della sez. 1.1.1:

$$T_{s\text{-}id\text{-}0} = N\ \tau$$

Il tempo di servizio ideale $T_{s\text{-}id}$ vale $N\tau$ se

$$N\ \tau \geq 2\,(\tau + T_{tr})$$

altrimenti vale $2(\tau + T_{tr})$.

1.1.4 Sistemi complessi e sottosistemi paralleli

Tutti i concetti finora discussi si estendono a sistemi complessi contenenti sottosistemi ognuno dei quali può consistere di una singola unità o di più unità operanti in parallelo.

Dato un sistema Σ operante su stream, composto da più unità, il suo **tempo di servizio ideale** è uguale all'inverso della *banda (frequenza) di interarrivo* B_A dello stream (degli stream) d'ingresso. Nel caso di un sistema "chiuso" (che genera lo stream al suo interno), tale frequenza è quella di *generazione dello stream* (degli stream).

Il **tempo di servizio effettivo** di Σ è uguale all'inverso della sua *banda (frequenza) di interpartenza* B_p, eventualmente valutata assumendo l'esistenza di stream di uscita fittizi quando il sistema sia "chiuso".

L'efficienza relativa dell'intero sistema è quindi data da:

$$\epsilon_\Sigma = \frac{B_p}{B_A}$$

Per ogni unità appartenente a Σ, il tempo di servizio ideale ed effettivo e l'efficienza relativa si determinano come detto in precedenza.

Se una singola unità U, avente tempo di servizio ideale $T_{s-id}^{(1)}$ viene trasformata in una struttura parallela con *grado di parallelismo n*, il tempo di servizio ideale di tale sottosistema è dato da:

$$T_{s-id}^{(n)} = \frac{T_{s-id}^{(1)}}{n}$$

Questa definizione si applica ricorsivamente: replicando n_2 volte un sottosistema, avente inizialmente grado di parallelismo n_1, si ottiene un nuovo sottosistema con grado di parallelismo complessivo $n = n_1\, n_2$:

$$T_{s-id}^{(n_1\, n_2)} = \frac{T_{s-id}^{(n_1)}}{n_2}$$

Il **tempo di completamento** per elaborare uno stream di *m* elementi viene valutato mediante la relazione (importantissima dal punto di vista dell'impatto tecnologico):

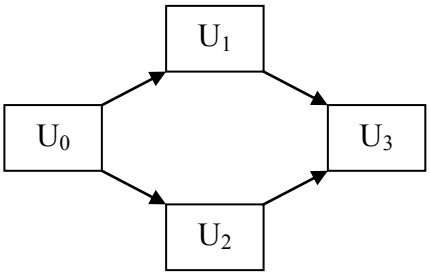
$$T_c^{(n)} \sim m\, T_s^{(n)}$$

La relazione vale rigorosamente per $n = 1$, mentre per *n* qualsiasi è valida con ottima approssimazione per $m >> n$.

1.2 Un esempio di sistema come grafo aciclico di unità

Consideriamo il seguente sistema Σ avente la struttura di un grafo aciclico "chiuso":

L'unità U_0 contiene una memoria A[N], applica ad ogni elemento di A una funzione F_0 e invia tale risultato a U_1 con probabilità p_1 e a U_2 con probabilità p_2, con $p_1 + p_2 = 1$. Quindi, U_0 genera uno stream di lunghezza $m = N$, alcuni elementi del quale sono inviati a U_1 e gli altri a U_2.



U_1 (U_2) esegue, su ogni elemento ricevuto, una funzione F_1 (F_2) e invia il risultato a U_3. U_3 riceve nondeterministicamente un valore da U_1 o da U_2, vi applica una funzione F_3 e scrive i risultati in una sua memoria B[N]. Se vogliamo garantire che i valori in B rispettino lo stesso ordine di A, ogni valore prodotto da U_0 è accompagnato dall'indice di A, e tale indice è propagato attraverso U_1 e U_2 fino a U_3.

I tempi di elaborazione delle funzioni F_0, F_1, F_2, F_3 siano rispettivamente 50τ, 20τ, 70τ, 40τ. Inoltre sia p_1 = 1/3 e p_2 = 2/3. La latenza di trasmissione di ogni collegamento sia $T_{tr} = 4\tau$.

Vogliamo determinare tempo di servizio effettivo, efficienza relativa e tempo di completamento di Σ, e tempo di servizio effettivo ed efficienza relativa per ognuna delle unità U_0, U_1, U_2, U_3.

1.2.1 Risoluzione del grafo

Tutte le comunicazioni hanno latenza

$$L_{com} = 2\,(t + T_{tr}) = 10\,\tau$$

quindi sono completamente mascherate dal calcolo in ognuna delle unità. Nell'elaborazione su stream, i tempi di servizio ideali sono quindi:

$$T_{0-id} = T_{F0} = 50\,\tau \quad T_{1-id} = T_{F1} = 20\,\tau \quad T_{2-id} = T_{F2} = 80\,\tau \quad T_{3-id} = T_{F3} = 40\,\tau$$

Il tempo di interpartenza di U_0 è uguale al suo tempo di servizio ideale:

$$T_{p0} = T_{0-id} = 50\,\tau$$

Questo è anche l'inverso della *banda di generazione dello stream* di tutta la computazione, quindi esprime il tempo di servizio ideale di Σ:

$$T_{\Sigma-id} = 50\,\tau$$

Secondo il *teorema del partizionamento dei serventi* (Cap. X, sez. 4.2.3) i tempi di interarrivo a U_1 e U_2 sono rispettivamente:

$$T_{\mathrm{A}1} = \frac{T_{p0}}{p_1} = 150\,\tau \qquad T_{\mathrm{A}2} = \frac{T_{p0}}{p_2} = 75\,\tau$$

Si noti il significato di questo teorema: la frazione p_1 della banda di interpartenza di U_0 costituisce la banda di interarrivo a U_1, la frazione p_2 della banda di interpartenza di U_0 costituisce la banda di interarrivo a U_2.

I fattori di utilizzazione delle unità U_1 e U_2 sono rispettivamente:

$$\rho_1 = \frac{T_{1-id}}{T_{\mathrm{A}1}} < 1 \qquad \rho_{21} = \frac{T_{2-id}}{T_{\mathrm{A}2}} < 1$$

Quindi nessuna delle due unità è collo di bottiglia. I loro tempi di servizio effettivi, uguali ai loro tempi di interpartenza, sono rispettivamente:

$$T_{p1} = \max(T_{1-id}, T_{\mathrm{A}1}) = 150\,\tau \qquad T_{p2} = \max(T_{2-id}, T_{\mathrm{A}2}) = 75\,\tau$$

Il tempo di interarrivo a U_3 si determina in base al *teorema dei clienti multipli* (Cap. X. Sex. 4.2.2), secondo il quale la banda di interarrivo complessiva al servente è uguale alla somma delle singole bande di interarrivo dai clienti (da cui il risultato fondamentale: <u>la banda di un sistema di *n* moduli indipendenti è uguale alla somma delle singole bande</u>):

$$\frac{1}{T_{\mathrm{A}3}} = \frac{1}{T_{p1}} + \frac{1}{T_{p2}}$$

Quindi:

$$T_{A3} = 50\,\tau$$

$$\rho_3 = \frac{T_{3-id}}{T_{A3}} < 1$$

$$T_{p3} = \max(T_{3-id}, T_{A3}) = 50\,\tau$$

Il tempo di interpartenza (su uno stream fittizio) da U_3 esprime il tempo di servizio effettivo di Σ, quindi:

$$T_\Sigma = 50\,\tau$$

$$\varepsilon_\Sigma = \frac{T_{\Sigma-id}}{T_\Sigma} = 1$$

Abbiamo così verificato una proprietà fondamentale dei sistemi descrivibili da *grafi aciclici*: *se nessuna delle unità componenti è collo di bottiglia, il tempo di servizio effettivo del sistema è uguale al suo tempo di servizio ideale, quindi la sua efficienza relativa è uguale a uno*.

Questo risultato ottimale si raggiunge, in questo come in altri casi, "sottoutilizzando" alcune delle unità componenti:

$$U_0 : T_{0-id} = 50\,\tau, T_0 = 50\,\tau, \varepsilon_0 = 1$$

$$U_1 : T_{1-id} = 20\,\tau, T_1 = 150\,\tau, \varepsilon_1 = \rho_1 = 0.13$$

$$U_2 : T_{2-id} = 70\,\tau, T_2 = 75\,\tau, \varepsilon_2 = \rho_2 = 0.93$$

$$U_3 : T_{3-id} = 40\,\tau, T_3 = 50\,\tau, \varepsilon_3 = \rho_3 = 0.8$$

Il risultato di raggiungere le prestazioni ottimali e, al tempo stesso, avere tutte le unità sfruttate al 100% si avrebbe se tutti i fattori di utilizzazione fossero esattamente uguali a uno. Ad esempio, le unità U_1, U_2, U_3 potrebbero essere progettate con tempi di servizio ideali più alti, senza conseguenze sulla banda complessiva fintanto che i fattori di utilizzazione si mantengono minori o uguali a uno.

In un sistema con struttura a grafo aciclico senza colli di bottiglia, la riduzione del tempo di servizio di singole unità non ha come conseguenza un aumento di *banda* complessiva, ma solo una diminuzione della *latenza*. Nel nostro caso, la latenza delle singole unità è data dal loro tempo di servizio ideale. Per Σ:

- attraversando il sottografo U_0, U_1, U_3 la latenza è data da:

$$L_a = L_0 + L_1 + L_3 + 2\,T_{tr} = 118\,\tau$$

- attraversando il sottografo U_0, U_2, U_3 la latenza è data da:

$$L_b = L_0 + L_2 + L_3 + 2\,T_{tr} = 168\,\tau$$

La latenza media del sistema è quindi:

$$L_\Sigma = p_1\,L_a + p_2\,L_b = 151.3\,\tau$$

Il tempo di completamento è dato da:

$$T_{c-\Sigma} \sim N\,T_\Sigma = 50\,N\,\tau$$

1.2.2 Valutazione in presenza di colli di bottiglia

Modifichiamo le specifiche di partenza, supponendo che il tempo di servizio ideale di U_1 sia ora $T_{1\text{-}id} = 540\ \tau$, con la conseguenza che U_1 diviene collo di bottiglia. Il suo tempo di interpartenza è uguale a 540 τ.

L'esistenza del collo di bottiglia provoca, *a regime*, una diminuzione della banda effettiva di U_0, in quanto il tempo di interarrivo a U_1 non è più

$$T_{\mathrm{A}1} = \frac{T_{p0}}{p_1} = 150\ \tau$$

bensì diviene uguale a $T_{1\text{-}id}$. Quindi il tempo di servizio effettivo di U_0 assume un valore T_0 tale che:

$$T_{\mathrm{A}1} = \frac{T_0}{p_1} = T_{1-id}$$

da cui:

$$T_0 = p_1\, T_{1-id} = 180\ \tau$$

L'esistenza del collo di bottiglia in U_1 *si ripercuote anche sul tempo di interarrivo a* U_2 (si noti che, in assenza di colli di bottiglia, i comportamenti temporali di U_1 e U_2 sono invece completamente indipendenti):

$$T_{\mathrm{A}1} = \frac{T_0}{p_2} = 270\ \tau$$

Il tempo di interarrivo a U_3 diviene uguale a 180 τ. Non essendo U_3 collo di bottiglia, questo è anche il valore del tempo di servizio effettivo di Σ:

$$T_\Sigma = 180\ \tau$$

Essendo ancora:

$$T_{\Sigma-id} = 50\ \tau$$

si ha

$$\varepsilon_\Sigma = \frac{T_{\Sigma-id}}{T_\Sigma} = 0.28$$

Il tempo di completamento diviene uguale a 180 N τ.

1.2.3 Parallelizzazione di colli di bottiglia: soluzione farm

Volendo eliminare, o quanto meno ridurre, gli effetti del collo di bottiglia, possiamo tentare di parallelizzare U_1.

A questo scopo, la metodologia di parallelizzazione prevede alcune *forme di parallelismo* standard, tutte caratterizzate da un modello di implementazione e un modello dei costi noto. Inoltre, sotto ampie condizioni le forme sono componibili tra loro. La trattazione del Cap. X descrive la forme di parallelismo pipeline, farm, data-flow, partizionamento funzionale, data-parallel.

La forma di parallelismo *data-parallel*, che si rivela molto potente in molte applicazioni pratiche a livello di processi, esula dagli scopi di questo corso nella sua accezione più

generale, e verrà usata solo in particolari casi semplici, seppur importanti (memoria modulare).

Le altre forme verranno utilizzate nello studio del parallelismo a livello di istruzioni.

La forma *pipeline* è stata esemplificata nella sez. 1.1.1. Esempi significativi saranno trattati nelle CPU con ILP.

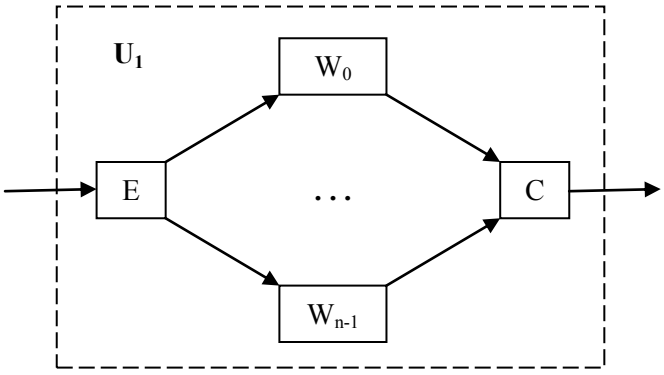
Un esempio di forma con *partizionamento funzionale* è rappresentato da una parte del presente esempio: il sottosistema $\{U_1, U_2\}$ può essere interpretato come una semplice parallelizzazione della computazione

$$if\ pred\ (x)\ then\ F_1(x)\ else\ F_2(x)$$

dove *x* è il generico valore prodotto da U_0. Se nessuna delle componenti (U_1, U_2 nell'esempio) è collo di bottiglia, il sistema con partizionamento funzionale è caratterizzato da bilanciamento del carico e la banda effettiva è uguale alla banda ideale, altrimenti il carico risulta sbilanciato e le banda effettiva risulta degradata.

In questo esempio la soluzione più immediata e più potente è il *farm*. Questa forma è applicabile quando la computazione da parallelizzare, operante *su stream*, sia espressa da una *funzione pura*, come nel caso di U_1. Tra l'altro, non è necessario conoscere l'implementazione interna di tale funzione, ma solo il suo comportamento all'interfaccia e il suo tempo di servizio ideale.

La soluzione farm consiste nel replicare U_1 in un certo numero *n* di copie identiche (*worker*), alle quali una unità *Emettitore* smista opportunamente i valori dello stream d'ingresso, e dalle quali una unità *Collettore* riceve i risultati da inviare sullo stream di uscita:



Il numero ottimo *n* di worker è tale da eliminare il collo di bottiglia, cioè tale che:

$$\rho_1 = \frac{T_1^{(n)}}{T_{A1}} = \frac{\frac{T_1^{(1)}}{n}}{T_{A1}} = 1$$

da cui, per ottenere un valore intero:

$$n = \left\lceil \frac{T_1^{(1)}}{T_{A1}} \right\rceil = \left\lceil \frac{T_{1-id}}{T_{A1}} \right\rceil = 4$$

Con questo grado di parallelismo, il *tempo di servizio ideale* del nuovo sottosistema $U_1 = \{E, W_0, W_1, W_2, W_3, C)$ vale:

$$T_{1-id}^{(4)} = \frac{T_1^{(1)}}{n} = 135\ \tau$$

Il *tempo di servizio effettivo* è uguale al tempo di interarrivo

$$T_1^{(4)} = T_{A1-id} = 150\ \tau$$

quindi ottimale, e l'efficienza relativa di U_1 nel suo complesso è la massima possibile (compatibilmente con la necessità di arrotondare *n* all'intero superiore):

$$\epsilon_1^{(4)} = \frac{T_{1-id}^{(4)}}{T_1^{(4)}} = 0.9 = \rho_1$$

Se il tempo di calcolo (funzione F_1) ha una varianza significativa rispetto alla media, le prestazioni ottimali si raggiungono a condizione di mantenere *bilanciato il carico* dei worker. Questo si può ottenere con un'implementazione dell'emettitore (scheduling dei task) *su domanda*: un nuovo valore in ingresso è inviato da E a uno dei worker che si dichiarino disponibili ad elaborarlo.

A livello firmware questa strategia è molto semplice ed efficiente: E è descritto da un microprogramma di una sola microistruzione nella quale vengono testati l'indicatore RDY dello stream d'ingresso e tutti gli ACK dai worker, instradando il valore d'ingresso ad una qualsiasi interfaccia di uscita avente ACK = 1. Il tempo di servizio ideale di questa unità è uguale a 1τ, per cui il suo tempo di servizio effettivo è uguale a L_{com}, nel nostro caso 10τ. L'unità E ha dunque bassa efficienza relativa (nell'esempio uguale a 0.1), ma è ben lungi dal rappresentare un collo di bottiglia. Le stesse prestazioni sono proprie del collettore C.

In conclusione, la soluzione farm realizzata offre una banda uguale o superiore alla banda richiesta.

In queste condizioni, il sistema Σ, che inizialmente aveva grado di parallelismo uguale a 4, può essere trasformato in una versione equivalente avente grado di parallelismo uguale a 9 (U_1 è sostituito da un sottosistema farm composto da 6 unità), riuscendo così ad ottenere le massime prestazioni:

$$T_\Sigma^{(9)} = 50\,\tau$$

$$\varepsilon_\Sigma^{(9)} = \frac{T_{\Sigma-id}}{T_\Sigma^{(9)}} = 1$$

Le prestazioni delle singole unità si valutano come visto in precedenza. Ogni worker ha tempo di servizio effettivo ideale uguale a 540τ, tempo di servizio effettivo uguale a 600τ, quindi efficienza relativa uguale a 0.9.

Infine, in una parallelizzazione tipo farm all'aumento di banda corrisponde un *aumento della latenza* rispetto al caso non parallelo, a causa della presenza dell'emettitore, del collettore e di canali ulteriori. Analogamente avviene con le forma di parallelismo pipeline e partizionamento funzionale. *L'impatto della latenza è significativo in computazioni a grafo ciclico*, tipicamente cliente-servente a domanda-risposta (sez. 3).

Invece le forme di parallelismo data-flow e data-parallel sono anche potenzialmente in grado di abbassare la latenza rispetto al caso non parallelo.

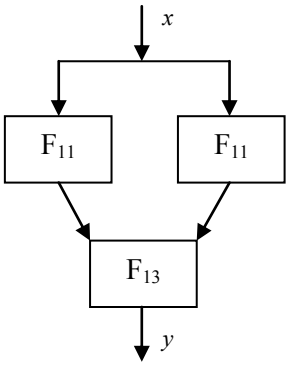
1.2.4 Parallelismo data-flow

Per concludere l'esempio, studiamo una diversa parallelizzazione di U_1, assumendo di conoscere la forma interna di F_1, ad esempio:

$$y = F_1(x) = F_{13}(F_{11}(x), F_{12}(x))$$

Supponiamo le funzioni F_{11}, F_{12}, F_{13} abbiano tempo di servizio identico e uguale a 180 τ.

Dalla computazione sequenziale è possibile ricavare un *grafo di ordinamento parziale* delle funzioni componenti (grafo data-flow), applicando le *condizioni di Bernstein* (Cap. IV, sez. 2.2.2) per ricavare le **dipendenze sui dati** tra le componenti stesse.



Nell'esempio, le funzioni F_{11} e F_{12} sono indipendenti, mentre F_{13} ha dipendenze sui dati da F_{11} e da F_{12}.

Ne discende il grafo *a logica AND* di figura, che rappresenta la versione data-flow di U_1 con grado di parallelismo uguale a 3.

Il tempo di servizio ideale di un sistema rappresentato da un grafo AND è dato dal massimo dei tempi di servizio delle unità componenti, quindi nel nostro caso:

$$T_{1-id}^{(3)} = 180\,\tau$$

Di conseguenza, con questa parallelizzazione U_1 rimane collo di bottiglia (tempo di interarrivo nel transitorio uguale a 150 τ), a conferma che, in presenza di computazioni che siano funzioni pure, la conoscenza della struttura interna della funzione non porta necessariamente vantaggio dal punto di vista dello studio della banda.

Si noti invece che, rispetto alla versione sequenziale e alla versione farm, si ottiene una minore *latenza*.

La determinazione del tempo di servizio effettivo e dell'efficienza relativa del sistema Σ, e delle singole unità, è lasciata come esercizio.

Ovviamente, essendo i componenti di questo grafo data-flow funzioni pure, sarebbe possibile parallelizzare i singoli nodi mediante il paradigma farm, ottenendo di nuovo le prestazioni massime. In generale, è agevole dimostrare che, quando la computazione complessiva sia una funzione pura, la soluzione farm è caratterizzata dalla massima banda con grado di parallelismo minore o uguale alla soluzione in cui più componenti di una forma di parallelismo diversa (pipeline, data-flow) siano a loro volte parallelizzate come farm.

L'esempio ci è servito soprattutto per evidenziare i concetti di computazione data-flow e di dipendenze sui dati, che verranno usati intensivamente nella trattazione del parallelismo a livello di istruzioni.

Quando la computazione complessiva non sia una funzione pura, la soluzione farm non è applicabile, mentre le soluzioni pipeline, data-flow e data-parallel sotto certe condizioni (partizionamento opportuno dello stato) sono applicabili.

2 Applicazione al sottosistema di memoria

2.1 Studio delle prestazioni di una memoria interallacciata

Consideriamo una unità di memoria *M* contenente un componente logico memoria con *m* moduli e organizzazione interallacciata degli indirizzi. Le operazioni esterne definite siano: 1) lettura di un blocco di *m* parole a indirizzi consecutivi, 2) scrittura di una singola parola.

Il tempo di servizio ideale vale:

$$T_{s-id}^{(1)} = \tau_M$$

in quanto questo è il tempo di servizio per entrambe le operazioni esterne.

Il tempo di completamento per uno stream di *n* richieste di scrittura è dato da:

$$T_{c-write\ -id}^{(1)} = n\,\tau_M$$

Consideriamo ora una trasformazione parallela di *M* in una struttura con *m* unità indipendenti: $M = \{M_0, \ldots, M_{m-1}\}$, tutte capaci di eseguire entrambe le operazioni esterne e con la generica M_i contenente nella propria PO solo il modulo *i*-esimo del componente logico memoria. Si tratta di una semplice applicazione della forma *data-parallel* (funzioni replicate e stato partizionato).

Facendo pervenire ogni richiesta contemporaneamente a tutte le unità (eventualmente attraverso una unità di interfaccia), il tempo di servizio ideale per la lettura blocco rimane invariato, in quanto la stessa versione sequenziale di *M* possedeva già il necessario grado di parallelismo al suo interno. In altre parole, la banda in lettura offerta, tanto da *M* come singola unità quanto da *M* come *m* unità, è uguale a un blocco (*m* parole) ogni τ_M. Questo esempio mostra come, *in talune computazioni*, una implementazione *a livello firmware* come singola unità possa raggiungere la stessa banda di una implementazione parallela, grazie alla peculiarità del livello firmware di permettere parallelismo a livello di ciclo di clock.

Il *minimo* tempo di servizio ideale per operazioni di scrittura singola vale:

$$T_{s-write\ -id-min}^{(m)} = \frac{\tau_M}{m}$$

in quanto la banda *massima* offerta da Σ si ha quando, in presenza di uno stream di richieste, *m* richieste consecutive si riferiscono a moduli distinti:

$$B_{id-max}^{(m)} = \frac{m}{\tau_M}$$

Di conseguenza, il tempo di completamento ideale per uno stream di *n* operazioni di scrittura *a indirizzi consecutivi* è dato da:

$$T_{c-write\ -id}^{(m)} \sim n\,T_{s-write\ -id-min}^{(m)} = \frac{n}{m}\tau_M$$

con una *scalabilità* rispetto al caso sequenziale uguale a *m* nell'ipotesi più favorevole che le richieste siano a indirizzi consecutivi.

Nel caso più generale che le operazioni richieste *non* siano a indirizzi consecutivi, si dimostra (vedi dispensa su ILP, sez. 3.3.3) che la banda è approssimabile come

$$B_M \sim \frac{m^{0.56}}{\tau_M}$$

nell'ipotesi, statisticamente attendibile, che gli indirizzi delle richieste siano distribuiti casualmente rispetto ai moduli di memoria (probabilità di accedere ad un qualsiasi modulo uguale a *1/m*).

Quindi, quando le richieste dello stream contengano indirizzi qualsiasi, la scalabilità di una memoria interallacciata è vicina a $\sqrt{m}$.

2.2 Scritture in memoria sincrone e asincrone

Nel calcolatore elementare studiato nel Cap. VI abbiamo supposto che, nell'elaborazione di una istruzione di STORE, il processore attenda esplicitamente la conclusione della scrittura in memoria. Questo funzionamento corrisponde a considerare una semantica *sincrona* dell'operazione di scrittura.

2.2.1 STORE asincrona

A differenza della lettura, per una scrittura è possibile prevedere anche una semantica *asincrona*: una volta inviata la richiesta, l'unità richiedente prosegue l'elaborazione senza attendere la fine della scrittura in memoria. Nel caso di un processore, questa semantica comporta l'attesa dell'esito dalla MMU relativamente alle sole operazioni effettuate dalla MMU stessa (controllo della condizione di fault di pagina e controllo di protezione), ma non dell'esito dell'operazione effettuata dalla memoria esterna. Eventuali eccezioni nel funzionamento della memoria esterna verranno rilevate successivamente e trattate facendo riferimento all'istruzione che le ha generate.

Nel caso di scrittura asincrona, il *tempo di servizio della fase di esecuzione* di una istruzione di STORE per un processore elementare diviene semplicemente:

$$T_{ex-STORE} = 3\,\tau$$

con ovvie implicazioni positive sul tempo di completamento.

Concettualmente, siamo in una situazione in cui CPU e memoria esterna operano (parzialmente) in parallelo. La CPU produce uno *stream di richieste di scrittura* verso la memoria (in termini modellistici, le richieste di lettura non vengono considerate appartenenti ad uno stream, in quanto eseguite a domanda e risposta senza parallelismo). Di conseguenza siamo nella situazione studiata nella sez. 1: affinché il tempo di servizio $T_{ex\text{-}STORE}$ sia valutabile come 3τ è necessario che sia soddisfatta la condizione:

$$T_p \geq \frac{1}{B_M}$$

dove T_p è il tempo di interpartenza dal processore delle richieste di scrittura, e B_M la banda della memoria, uguale a *1/ta* per una memoria sequenziale oppure valutata come nella sez. 2.1 per una memoria interallacciata.

Se la condizione suddetta non è soddisfatta, T_p deve essere sostituito da $1/B_M$ nella valutazione del tempo di completamento.

2.2.2 Impatto tecnologico della STORE asincrona nei processori esistenti

Praticamente tutti i processori esistenti prevedono la STORE con semantica asincrona, ma soli alcuni prevedono tanto la semantica sincrona quanto quella asincrona (usando codici operativi diversi oppure un'annotazione). Si parla di processori con **Weak Store Ordering** (WSO) quando l'unica semantica sia quella asincrona, e di processori con **Total Store Ordering** (TSO) quando sia prevista anche la semantica sincrona:

- nelle macchine WSO l'ordinamento tra accessi in memoria, rispetto all'ordine con cui compaiono nel programma le istruzioni LOAD/STORE che li provocano, non è garantito in maniera primitiva dall'architettura firmware;
- nelle macchine TSO l'utilizzo della STORE sincrona garantisce sempre l'ordinamento suddetto, per cui il ricorso alla STORE sincrona può risolvere casi critici nei quali l'ordinamento tra operazioni in memoria influenzi la corretta semantica del programma.

Il problema dell'ordinamento delle operazioni in memoria (memory ordering) è sentito soprattutto nelle architetture multiprocessor, nelle quali, a causa di ritardi variabili e conflitti nella struttura che interconnette processori a memorie esterne, l'ordine con cui un processore osserva il risultato di operazioni in memoria può essere diverso da quello con cui le operazioni sono state richieste da altri processori. Ma anche in un sistema uniprocessor il problema può sorgere a causa del parallelismo tra CPU e sottosistema di I/O, sia per accessi in Memory Mapped I/O che in DMA. Analoghi problemi potranno emergere nella trattazione del parallelismo a livello di istruzioni (ILP) per processori "out-of-order".

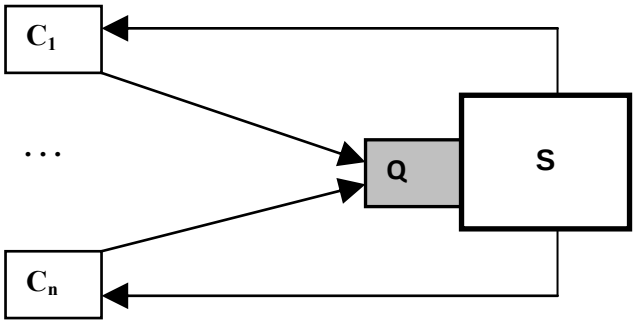
Per far fronte a problemi di consistenza delle informazioni in memoria, i processori WSO prevedono una esplicita istruzione (detta *Memory_Barrier*, o *Fence*) che forza l'attesa della conclusione di tutte le istruzioni LOAD e STORE lanciate fino a quel momento. In pratica, tale istruzione rende sincrono l'insieme di tutte le STORE precedenti a un determinato punto critico del programma. Spesso anche processori TSO dispongono della Memory_Barrier da usare talvolta in alternativa alla STORE sincrona per cercare di sfruttare, ove possibile, la semantica asincrona per ovvie ragioni di prestazioni.

Accenniamo infine al fatto che le Memory_Barrier non vanno confuse (come talvolta accade nella documentazione di linguaggi e compilatori) con le così dette Compiler Barrier (e/o con la keyword `volatile` in C e C++) il cui scopo è di forzare l'ordinamento tra specifiche istruzioni che (come vedremo trattando le architetture ILP) un compilatore ottimizzante potrebbe alterare esclusivamente per ragioni di prestazioni, ma senza implicazioni sulla correttezza del programma.

3 Modello cliente-servente a domanda-risposta

Per quanto limitino potenzialmente il parallelismo di un sistema, le interazioni a domanda-risposta sono talvolta inevitabili per ragioni legate alla strutturazione della computazione.

Il modello cliente-servente a domanda-risposta è studiato nel Cap. X, sez. 12, in termini di concetti basici di teoria delle code.



In questa sede verranno evidenziati i concetti chiave di questo importante argomento.

Il modello astratto è applicabile quando effettivamente esistano più unità clienti (con collegamenti dedicati o ripartiti verso il servente), ma anche quando esista un solo cliente purché capace di generare uno *stream di richieste*, ad esempio quando il cliente sia parallelo al suo interno e/o, comunque, quando non si limiti ad attendere la risposta a una richiesta ma sia capace di proseguire l'elaborazione inviando eventualmente altre richieste.

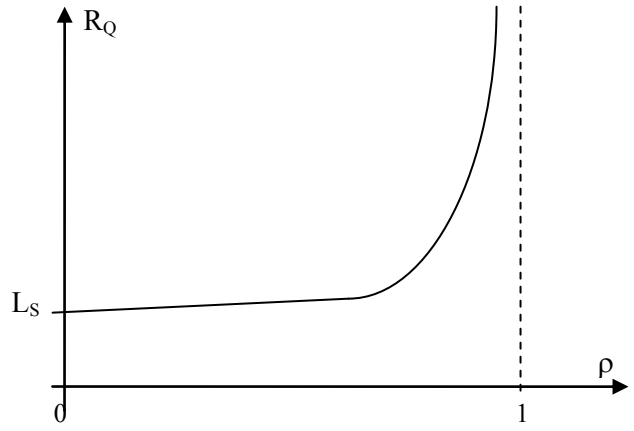
Un sistema con un singolo cliente che per ogni richiesta attenda la relativa risposta, senza avere parallelismo tra cliente e servente, rappresenta un caso particolare degenere nel quale non si ha formazione di coda. In questo caso l'analisi delle prestazioni dipende esclusivamente dalla latenza del servente.

L'aspetto fondamentale del modello è la valutazione del **tempo di risposta** del servente, che in generale è esprimibile come:

$$R_Q = W_Q(\rho) + L_S$$

dove il termine W_Q è il ***tempo medio di permanenza di una richiesta in coda*** e L_S la ***latenza del servente***.

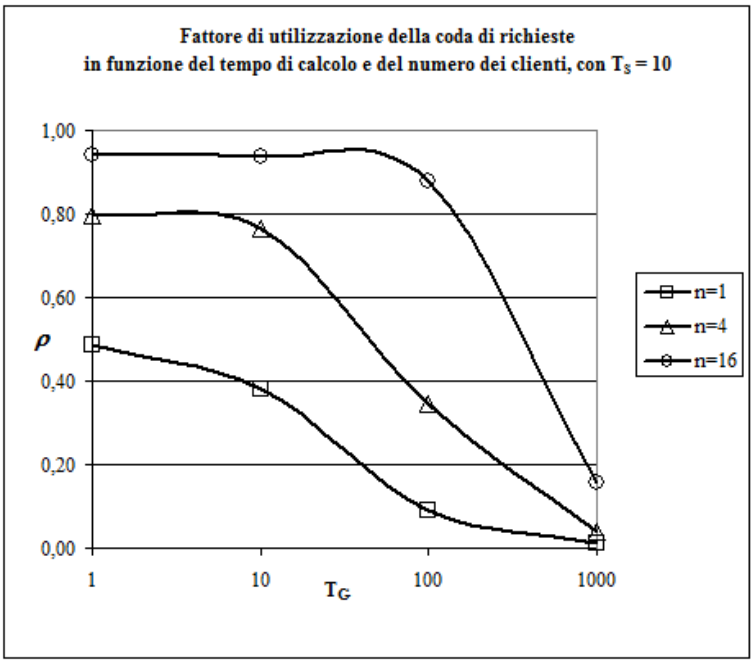
Poiché W_Q è una funzione del fattore di utilizzazione (la forma della funzione dipende dalle distribuzioni probabilistiche degli interarrivi e dei servizi: Cap. X, sez. 12), fissato il tempo di servizio ideale e il numero di clienti W_Q dipende dal tempo di servizio del servente. Quindi, la relazione precedente ci permette di *scorporare gli effetti della banda del servente e della sua latenza sul tempo di risposta*.



L'andamento qualitativo di R_Q in funzione di ρ è monotono crescente, con $R_Q = L_S$ per ρ = 0 e R_Q tendente all'infinito per ρ tendente a uno.

L'effetto autostabilizzante del funzionamento a domanda-risposta è tale per cui è sempre $\rho < 1$.

Gli elementi che contribuiscono al valore di ρ, e quindi al tempo di risposta, sono il tempo di servizio ideale del cliente, quello del servente e il numero di clienti.



La figura mostra un esempio di andamento del fattore di utilizzazione al variare di *Tcl-id* (T_G in figura) e *n*, con T_s costante (nell'ipotesi di distribuzioni esponenziali del tempo di interarrivo e del tempo di servizio)

Il tempo di servizio effettivo del generico cliente è dato da:

$$T_{cl} = T_{cl-id} + R_Q$$

Tale valore è limitato inferiormente da $T_{cl\text{-}id} + L_S$ per ρ = 0, e tende all'infinito al tendere di ρ a uno.

In conclusione, nella progettazione di un servente è importante cercare di minimizzare *tanto* il suo tempo di servizio (attraverso il parallelismo) *quanto* la sua latenza.

Nella trattazione del parallelismo a livello di istruzioni vedremo una importante applicazione di questi concetti.

4 Applicazione alla gerarchia di memoria con cache

4.1 Trasferimento di blocchi

L'implementazione del trasferimento dei blocchi di cache (Cap. VIII) è un esempio significativo di *relazione tra banda e latenza di un servente* in un sistema cliente-servente a domanda-risposta.

Consideriamo inizialmente un calcolatore elementare con processore e memoria a domanda-risposta senza parallelismo.

Come sappiamo, in una gerarchia *on-demand* per trarre vantaggio dalla latenza dei programmi, senza degrado di prestazioni, è fondamentale *minimizzare la latenza del trasferimento di un blocco*. Questo si ottiene in due modi:

1. ricorrendo a forme di parallelismo della memoria principale che concorrono *anche* a ridurre la latenza. Con una memoria interallacciata, nel trasferimento di blocchi *l'aumento di banda che si ottiene è funzionale alla diminuzione della latenza*, proprio perché i dati vengono trasferiti in parallelo. La riduzione di latenza attraverso l'aumento di banda è una proprietà generale della forma data-parallel (oltre che della forma data-flow);
2. ricorrendo a *tecnologie di memoria che riducano la latenza di per sé*, senza introdurre parallelismo. In pratica, questo principio conduce all'introduzione di ulteriori livelli di cache, in particolare la cache secondaria.

Combinando queste due soluzioni, si giunge alla soluzione architetturale più frequente, nella quale la cache primaria funziona *su domanda*, ma i livelli superiori di cache funzionano con *prefetching*, e le memorie esterne al chip CPU hanno organizzazione interallacciata.

Il **prefetching dei blocchi** sfrutta il parallelismo tra cliente e servente (cache secondaria e memoria principale) e l'aumento di banda del servente. Si tratta della situazione in cui un singolo cliente genera uno *stream* di richieste al servente. Di conseguenza, il prefetching non è affetto da degradazione significativa se viene *minimizzato il tempo medio di attesa in coda*. Ciò comporta *minimizzare il fattore di utilizzazione del servente*, e quindi:

a) minimizzare il tempo di servizio del servente, ciò che si ottiene attraverso un aumento di banda tipico dell'organizzazione interallacciata;

b) non aumentare oltre un certo limite la banda delle richieste da parte del cliente (frequenza del prefetching), per non rendere troppo basso il tempo di interarrivo alla coda.

4.2 Write-Through

Un'altra applicazione dei principi generali alle gerarchie di memoria con cache si ha nell'analisi delle prestazioni di sistemi con *Write-Through* (Cap. VIII). Ogni richiesta di scrittura viene propagata in parallelo (ad esempio da parte della MMU) alla cache primaria, alla cache secondaria (e alla cache terziaria, se esiste) e alla memoria principale.

La scrittura è necessariamente *asincrona nei confronti della memoria principale*, quindi siamo in presenza di una computazione a *grafo aciclico*.

Affinché il meccanismo del Write-Through non comporti una degradazione delle prestazioni occorre che (come visto nella sez. 2.2)

$$T_p \geq \frac{1}{B_M}$$

dove T_p è il tempo di interpartenza dal processore delle richieste di scrittura, e B_M la banda della memoria principale, valutata come nella sez. 2.1 per una memoria interallacciata.

Se la condizione suddetta non è soddisfatta, T_p deve essere sostituito da $1/B_M$ nella valutazione del tempo di completamento.

Ad esempio, consideriamo il seguente programma:

int A[N];

$\forall$ *i = 0 .. N–1:*

A[i] = F(A[i])

Sia T_{iter} il tempo medio di servizio di una generica iterazione. Poiché ad ogni iterazione avviene una scrittura, se

$$T_{iter} \geq \frac{1}{B_M}$$

allora il tempo di completamento è effettivamente dato da:

$$T_c = N\, T_{iter} + T_{fault}$$

altrimenti:

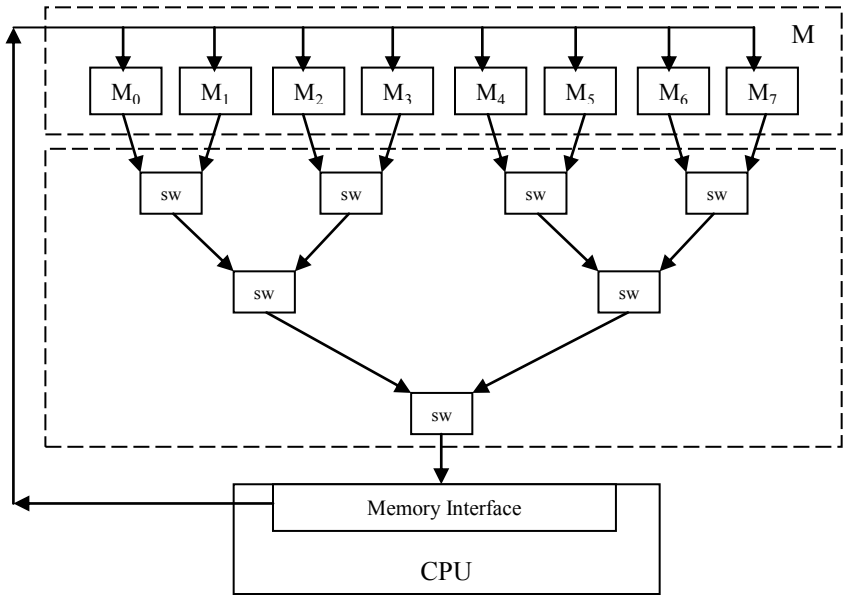
$$T_c = N\, \frac{1}{B_M} + T_{fault}$$

4.3 Interconnessione memoria – CPU per il trasferimento di blocchi

Nel cap. VIII si è supposto che il trasferimento di un blocco da una memoria interallacciata, con *m* moduli, alla CPU (alla cache secondaria o alla cache primaria) venga effettuato utilizzando *m* collegamenti distinti.

Questa soluzione minimizza la latenza dell'interconnessione, ma è raramente applicabile per ragioni di *pin-count* del chip CPU.

Per minimizzare il pin-count, e anche per non legare la struttura della CPU alla particolare implementazione della memoria esterna, l'interfaccia di memoria del chip CPU verso la memoria esterna consta di una *singola parola* in ingresso (più eventuali informazioni di controllo e di esito).



Questo comporta la necessità di trovare una struttura di interconnessione ad alta banda e con latenza limitata (anche se non la minima latenza in assoluto, che è possibile solo con *m* interfacce distinte).

Un'interessante struttura è quella di figura con topologia ad *albero.*

In figura è mostrato il caso di un albero binario con *m* foglie.

Tutti i collegamenti sono di una parola.

Alla lettura di un blocco di *m* parole, queste si propagano in parallelo-pipeline lungo l'albero.

Ogni *unità di switch* (SW) fa il merge dei due stream di parole ricevute dai collegamenti in ingresso, generando uno stream di lunghezza doppia sul collegamento in uscita.

Il tempo di interpartenza da ogni unità SW si dimezza ad ogni livello attraversato.

La *banda* di trasferimento del blocco vale ancora

$$B_M = B_{M-max} = \frac{m}{\tau_M}$$

senza alcuna degradazione rispetto al caso di *m* interfacce distinte.

Quella che cambia è la *latenza*, che da costante passa a logaritmica. Complessivamente la *latenza di trasferimento di un blocco* di $\sigma = m$ parole vale:

$$T_{trasf} = 2T_{tr} + \tau_M + lg_2 m\,(\tau + T_{tr})$$

Il termine

$$T_{hop} = \tau + T_{tr}$$

è la latenza di un passo di routing ("hop") attraverso la struttura ad albero; tale passo consiste nell'attraversamento di un'unità SW (latenza uguale a 1τ) e del suo collegamento in uscita (latenza uguale a T_{tr}).

Si noti che la scrittura in cache si sovrappone al trasferimento del blocco: infatti lo stream di parole ricevute sulla singola interfaccia della CPU, con tempo di interarrivo uguale a $1/B_M$, è scritto nella cache (secondaria o primaria) con tempo di servizio uguale a τ.

In generale, per σ multiplo di *m* si ha:

$$T_{trasf} = 2T_{tr} + \frac{\sigma}{m}\tau_M + lg_2 m\,(\tau + T_{tr})$$

in quanto ogni latenza $lg_2 m\,(\tau + T_{tr})$ per il trasferimento di *m* parole, tranne l'ultima volta, è sovrapposta all'intervallo τ_M per la lettura delle *m* parole successive.

Si confronti l'espressione ottenuta con quella valida per *m* interfacce distinte:

$$T_{trasf} = 2\,T_{tr} + \frac{\sigma}{m}\tau_M + m\,\tau$$

La differenza risulta molto contenuta per valori tipici di *m* e T_{tr}.

In conclusione, il notevole risparmio in pin-count e la generalità dell'interfaccia di memoria sono pagati con una penalità in latenza del tutto accettabile.

4.4 Cache e Memory Mapped I/O

La struttura di interconnessione che, tradizionalmente, viene usata in alternativa a quella della sezione precedente per minimizzare il pin-count è il *bus*. Come sappiamo, la banda di un bus è di un solo messaggio (1-2 parole) alla volta (per ciclo di clock del bus), in quanto non permette parallelismo, e la latenza è lineare in *n*. Dunque, si tratta di una struttura non adatta al trasferimento di blocchi in una gerarchia di memoria con cache.

Queste considerazioni fanno emergere chiaramente come, con strutture di interconnessione tra CPU e sottosistema di I/O aventi banda limitata (Bus di I/O), il trasferimento di blocchi da memorie di I/O alla cache della CPU sia caratterizzato da latenze incompatibili con una buona esplicitazione della località.

Nonostante gli standard per l'interconnessione di I/O (ad esempio, PCI) dichiarino bande apparentemente elevate, queste non possono di fatto essere raggiunte nel trasferimento di blocchi di cache. In effetti, queste strutture sono state sì concepite per il trasferimento ad alta banda di blocchi, ma relativamente a blocchi da trasferire tra memoria secondaria e memoria principale, nel qual caso la latenza si rivela sufficiente. Invece, alla luce della relazione che esiste tra banda e latenza per la gerarchia di memoria con cache, la banda di I/O non è sufficiente per minimizzare la latenza dei trasferimenti di blocchi di cache, che essenziale per sfruttare la località dei programmi.

Inoltre (o proprio a causa di quanto ora discusso) le memorie di I/O raramente sono realizzate a larga banda.

Per questi motivi, diverse macchine esistenti permettono di *disabilitare a programma* l'utilizzo della cache in presenza di accessi in Memory Mapped I/O.

Va da sé che strutture di interconnessione ad alta banda, come quelle della sezione precedente, permetterebbero un uso ben più intensivo del Memory Mapped I/O.