Gradi di parallelismo

E' possibile individuare i livelli architetturali nei quali si può applicare il concetto di parallelismo che in ultima istanza deve consentire al sistema un significativo aumento nelle prestazioni. I livelli possono essere distinti a partire dal basso della struttura elettronica e quindi:

L0: parallelismo su **microistruzioni** (tecnologia RISC)

L1: parallelismo su **istruzione** (tecniche di prefetch e di pipeline)

L2: parallelismo sul **sistema** (es. trasferimento operato dal DMA controller)

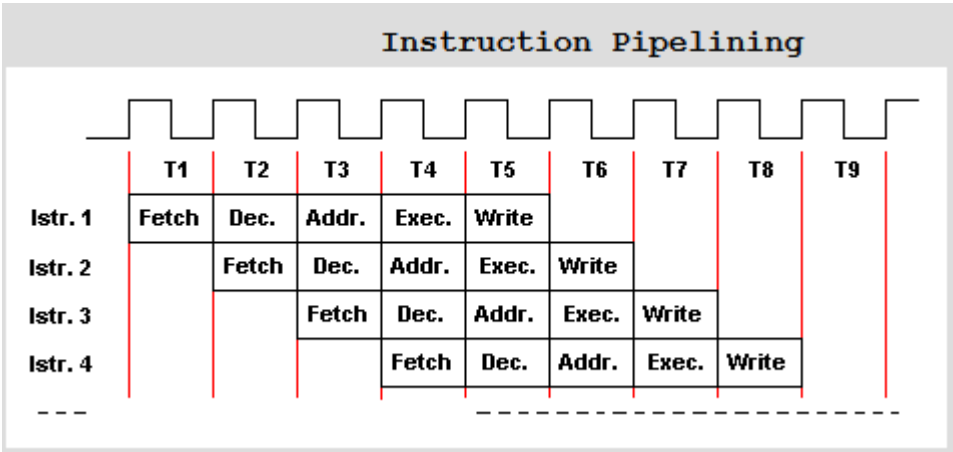
L3: parallelismo sul **processo** (multitasking (Windows 3.X) o in timesharing (UNIX))

L4: parallelismo interno al **programma** (macchine MIMD, algoritmi paralleli, SO distribuiti multiprocessor).

Evidentemente il parallelismo dei primi tre livelli implica un'evoluzione dell'hardware che non si riflette, almeno in un primo momento, sulla natura del sw che dovrà girarvi sopra.

A livello di microistruzioni, a loro volta inserite in microprogrammi associati ad ogni istruzione e utilizzate per espletare una qualsiasi delle fasi di processo di CPU (fetch, decode, operand fetch, execute, operand store), si realizza un grado alto di parallelismo data la scomponibilità di molte operazioni (es. FPU in parallelo, operazioni su IP e di calcolo). La tecnologia RISC (Reduced Instruction Set Computer), implementata parzialmente anche su macchine CISC (Complex Instruction Set Computer, ad es. sullo 80486) contribuisce all'ottimizzazione del codice a livello di microprogramma, riducendone e uniformandone i formati e gli aspetti operativi.

A livello di istruzione è evidente come le tecniche di prefetch e di pipeline hanno contribuito a parallelizzare il processo globale in CPU.



1)Fase di **Fetch**, che consiste nel caricare nel **Registro Istruzioni** della **CPU** il codice macchina della prossima istruzione da eseguire.
2) Fase di **Decodifica**, che consiste nel raccogliere una serie di informazioni riguardanti: il compito che l'istruzione richiede di svolgere alla **CPU**, la presenza di eventuali operandi, etc.
3) Fase di **Addressing** (indirizzamento), che consiste nel predisporre gli indirizzi per caricare gli eventuali operandi.
4) Fase di **Execute** che consiste nell'esecuzione vera e propria dell'istruzione da parte della **CPU**.
5) Fase di **Write Back** che consiste nel salvataggio del risultato nell'operando destinazione (registro o locazione di memoria).

Nel caso illustrato si parla di **pipeline a 5 stadi**; come si può notare, in questo caso la **CPU** può operare istante per istante su più istruzioni differenti. Più aumentano gli stadi della pipeline,

maggiore sarà il numero di istruzioni elaborate in contemporanea e minore sarà il tempo richiesto per ogni fase; in più bisogna anche considerare un ulteriore aumento delle prestazioni dovuto al fatto che spesso alcune delle fasi descritte prima non sono necessarie (caricamento degli operandi, write back).

Le questioni relative a modifiche del program counter ad esempio sono state affrontate predisponendo particolari dispositivi interni alla CPU in grado di **predire** statisticamente la modifica di IP tramite opportune tabelle storiche dei salti (BTB es. adottata sul PENTIUM); in questo caso le code di prefetch possono essere ricostruite in tempo reale e rese disponibili anche in caso di salto. La questione della dipendenza dei dati è stata affrontata predisponendo opportuni buffer interni che consentono la gestione intelligente del data-forwarding, potendo rilevare tempestivamente la dipendenza dei dati tra istruzioni concorrenti e quindi operare salvataggi temporanei che consentono l'avanzamento coerente del processo. Infine la questione dell'accesso concorrente a risorse comuni è stato ampiamente affrontato duplicando opportunamente gli accessi ad esempio al bus di sistema, predisponendo anche in questo caso buffer temporanei di memorizzazione.

Le **CPU** di classe **80586** e superiori, utilizzano due o più **pipeline** attraverso le quali possono eseguire in alcuni casi più istruzioni in contemporanea; questa situazione si verifica, ad esempio, quando la **CPU** incontra due istruzioni consecutive, indipendenti l'una dall'altra e tali da non provocare stalli nella **pipeline**. Se si verificano queste condizioni, è possibile eseguire le due istruzioni in parallelo; le **CPU** capaci di eseguire più istruzioni in parallelo vengono definite **superscalari**.
Una potentissima caratteristica delle **CPU** superscalari è data dalla possibilità di poter "indovinare" la direzione più probabile, lungo la quale proseguirà l'esecuzione di un programma; in sostanza, in presenza di una istruzione di salto "condizionato", la **CPU** può tentare di indovinare se il salto verrà effettuato o meno. Questa caratteristica viene definita **branch prediction** e permette in moltissimi casi di evitare lo svuotamento della pipeline in presenza di una istruzione di salto.
In pratica, per alimentare ciascuna pipeline vengono utilizzate due code di prefetch; la prima viene riempita nel solito modo (leggendo le istruzioni dalla memoria, una di seguito all'altra), mentre la seconda viene, invece, riempita in base alle "previsioni" fatte da un apposito algoritmo di branch prediction. Se nella prima coda viene incontrata una istruzione di salto e se l'algoritmo di branch prediction ha azzeccato le previsioni, allora la seconda coda conterrà le nuove istruzioni che ci servono; in questo modo, si evita lo svuotamento delle pipeline e l'esecuzione non subisce nessun rallentamento.

Il parallelismo di sistema invece fu brillantemente affrontato ad esempio nel caso di trasferimenti pesanti di I/O verso e per la memoria, trasferimenti che potevano essere eseguiti da dispositivi autonomi che si appropriano del bus di sistema consentendo alla CPU di avanzare in parallelo ad esempio in fase di microprogrammazione quando quei bus non sarebbero comunque utilizzati (es. trasferimento diretto in memoria operato dal DMA controller). Quasi tutte le architetture moderne sono coadiuvate da dispositivi autonomi e sincronizzati che sollevano la CPU da gravosi compiti: spesso queste collaborazioni avvengono su fasi parallele (es. interrupt controller quando riconosce l'interruzione, la accoda e la rende con l'id. direttamente alla CPU).

Il livello immediatamente superiore invece è assolutamente delegato al sw ed in particolare al sw di sistema operativo. Il parallelismo a livello di processo (task) è governato totalmente da uno specifico SO che ha la capacità di lavorare in multitasking (es. Windows 3.X) o in timesharing (es. UNIX). In realtà siccome condizione necessaria per SO del genere è la predisposizione dell'hw alla protezione della memoria per evitare conflitti tra i diversi task, ci sembra che anche da questo punto di vista l'hw debba essere tenuto doverosamente in considerazione. Il sw comunque garantirà la concorrenza e il parallelismo tramite l'opportuna duplicazione e virtualizzazione delle risorse hw più rilevanti come CPU, Memoria principale e alcune periferiche.

Il parallelismo a livello di programma infine costituisce una delle nuove frontiere del mondo informatico: si tratta di strutturare i programmi e gli algoritmi in essi contenuti in modo che la risoluzione sia affrontabile e risolvibile con più CPU. In questo caso si pongono numerose questioni a tutti i livelli e tutte imprescindibili che hanno, di fatto, ostacolato la diffusione di macchine parallele:

- strutturazione dell'hardware con n CPU e con una architettura di sincronizzazione specifica (macchine MIMD).
- strutturazione del software con metodologie di sviluppo completamente differenziate dalla norma (algoritmi paralleli).
- strutturazione del sistema operativo come coordinatore delle due unità hw e sw (SO distribuiti multiprocessor).

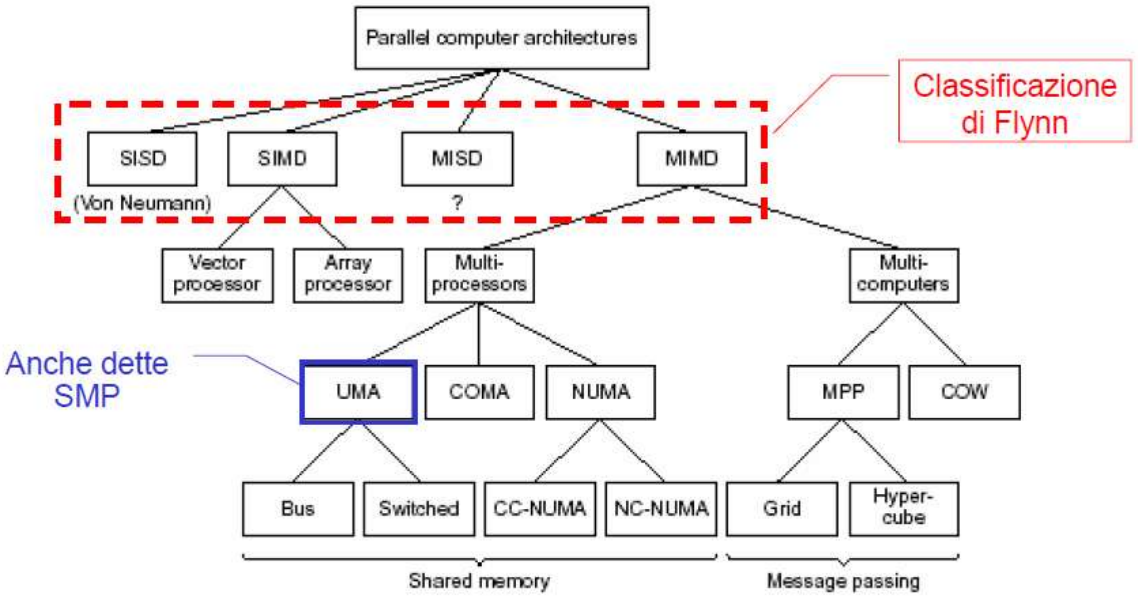
I supercomputer oggi: applicazioni e architettura
http://tweb.ing.unipi.it/tia/arch_par_slides.pdf ... progetti Blue Gene al 2012, generalità e futuro

Cluster di PC
http://it.wikipedia.org/wiki/Computer_cluster

Introduzione ai CLUSTER di PC
http://www.bitportal.it/tutorial/cluster.html

Tassonomia di Flynn
http://it.wikipedia.org/wiki/Tassonomia_di_Flynn

2009/2010
http://www.ce.uniroma2.it/courses/sd0910/lucidi/ClassArchPar.pdf



2002/2003
http://www.architettura.unina2.it/docenti/areaprivata/311/documenti/L04.pdf