Arquitectures multifil

Francesc Guim
Ivan Rodero

PID_00215406

Universitat Oberta de Catalunya

Índex

Introducció

Durant molts any el rendiment dels processadors ha anat incrementant, i ha explotat el nivell de paral·lelisme inherent al flux d'instruccions d'una aplicació. El 1971 es va llançar a la venda el primer microprocessador simple, l'anomenat Intel 4004. Aquest processador, de 4 bits, estava format per una sola unitat aritmeticològica: un banc de registres amb 16 entrades i un conjunt de 46 instruccions. El 1974, Intel va fabricar un nou microprocessador de propòsit general de 8 bits, el 8080, que contenia 4.500 transistors i era capaç d'executar un total de 200.000 instruccions per segon.

Des d'aquests primers processadors, en què tan sols es podien executar 200.000 instruccions per segon, s'ha arribat a arquitectures tipus Sandy Bridge (Intel) o AMDfusion (AMD) en què es poden arribar a executar aproximadament 2.300 bilions d'instruccions per segon. Per tal d'assolir aquest increment del rendiment, a grans trets es distingeixen dos tipus de millores importants aplicades als primers models esmentats anteriorment: tècniques associades a explotar el paral·lelisme de les instruccions i tècniques associades a explotar el paral·lelisme inherent de les aplicacions multifil.

El primer conjunt de tècniques va consistir a mirar d'explotar el nivell de paral·lelisme de les instruccions que componien una aplicació (*instruction level parallelism*, ILP, d'ara endavant). Exemples de processadors que les van usar són VAX 78032, PowerPC 601, els Intel Pentium o bé els AMD K5 o AMD K6.

Dins d'aquest primer grup de millores hi ha, entre d'altres: arquitectures superescalars, execució fora d'ordre d'instruccions i tècniques de predicció o *very long instruction word* (VLIW) (Fisher, 1893). Aquestes darreres optimitzacions eren força interessants des del punt de vista de l'ILP, atès que es basaven en la possibilitat que tenen els compiladors de millorar la planificació de les instruccions. D'aquesta manera el processador no necessita dur-les a terme.

El segon conjunt de tècniques intenten explotar el paral·lelisme associat als fils que componen les aplicacions. Durant les dècades del 1990-2010 la quantitat de fils que componen les aplicacions ha augmentat notòriament. Es va passar de l'ús de pocs fils d'execució a l'ús de milers de fils per aplicació. Un clar exemple d'aquest tipus d'aplicació són les emprades pels servidors, per exemple Apache o Tomcat. No obstant això, també trobem aplicacions multifil en els ordinadors domèstics. Exemples d'aquestes aplicacions són màquines virtuals com Parallels i VMWare o navegadors com Chrome o Firefox.

Per tal de donar suport a aquest tipus d'aplicacions, el maquinari ha anat fent l'evolució natural causada per aquesta demanda creixent de paral·lelisme. Com es veu a continuació, aquesta tendència es va fer palesa amb els primers processadors amb *simultaneous multithreading* i s'ha confirmat amb l'aparició de sistemes multinucli.

Objectius

Els principals objectius que ha d'assolir l'estudiant amb aquest mòdul didàctics són els següents:

1. Veure les limitacions inherents a les arquitectures de processador que exploten només el paral·lelisme a nivell d'instrucció.

2. Entendre quins són els beneficis d'usar arquitectures que exploten el paral·lelisme a nivell de fil.

3. Conèixer quin tipus d'arquitectures multifil hi ha i quines són les propietats de cadascuna.

4. Entendre com es pot millorar el rendiment dels processadors explotant el paral·lelisme a nivell de fil i a nivell d'instrucció alhora.

5. Saber en detall els diferents tipus d'arquitectures multifil i el disseny d'alguns processadors comercials que s'han dissenyat.

6. Entendre en detall les arquitectures multinucli i les diferents característiques que en poden fer variar més el disseny i rendiment.

1. Motivació

La tendència d'evolució del rendiment dels processadors és duplicar-se cada any respecte de l'any anterior. Aquest rendiment esdevé gràcies a dos factors: una millora del procés i una millora d'arquitectura. Aquest factors s'anomenen *tics* i *tocs*, respectivament. En les línies de processadors actuals (per exemple, la gamma de servidors d'Intel coneguda com *Xeon*; Rusu, Tam, Muljono, Ayers i Chang, 2006), aquests dos factors es van intercalant (un processador és un *tic* i el següent és un *toc*).

El primer factor és un factor tecnològic associat al procés de fabricació. Els transistors cada cop són més petits i això permet incrementar la mida dels recursos sense augmentar-ne la del processador (per exemple, duplicar la mida de la memòria cau o el nombre de bits que els busos poden transmetre). Aquest increment de capacitat es tradueix clarament en una millora de rendiment.

El segon pretén treure rendiment de canvis arquitectònics importants al processador (per exemple, passar d'una arquitectura en ordre a una arquitectura d'ordre). En el cas dels processadors ILP aquest segon factor acabaria sent un límit si no s'entrés en un altre tipus d'arquitectures en què s'explotessin també altres factors. La limitació principal és que el rendiment s'intenta explotar en un sol flux d'execució. Això per si mateix ja és una limitació important, perquè sovint els fluxos tenen característiques que per si mateixes ja limiten el rendiment que se'n pot treure (per exemple, errors de pàgines, errors per kiloinstruccions, quantitats de salts, dependències entre instruccions, etc.).

En qualsevol cas, l'escalabilitat del rendiment d'aquests sistemes és finita, de manera que per a obtenir una millora contínua en el rendiment dels processadors és bàsic optimitzar-ne l'arquitectura, però també és necessari millorar-ne el procés de fabricació. L'estudi de les tècniques de millora d'arquitectura són la motivació principal del mòdul que es presenta a continuació.

2. Preliminars: límits del paral·lelisme a nivell d'instrucció

Tal com s'esmenta en l'apartat anterior, durant les primeres dècades de desenvolupament de microprocessadors (1970 i 1980), l'objectiu principal es va centrar a explotar al màxim el paral·lelisme de les aplicacions a nivell d'instrucció. El límit d'aquest tipus de paral·lelisme és el que va donar lloc a una segon era de microprocessadors, en la qual el paral·lelisme es mou a nivell dels fils d'execució. En aquest apartat es presenta un estudi de límits en què es pretén demostrar que realment el rendiment de les tècniques ILP va assolir el seu llindar màxim, tenint en compte, és clar, el tipus d'aplicacions i ciència de computació que s'entén actualment.

Per tal de dur aquest estudi de límits, s'analitza l'impacte de córrer sis *benchmarks* (Bailey, Barton, Lasinski i H., 1991) molt emprats per a estudiar microprocessadors en una màquina ILP infinita, és a dir, una màquina de recursos infinits i perfectes. Les seves aplicacions es presenten en la taula 1 i s'expliquen amb més detall en Bailey, Barton, Lasinski i H. (1991). L'objectiu d'aquest estudi és, doncs, veure el resultat de córrer els *benchmarks* esmentats en una màquina ILP perfecta. Les característiques de la màquina perfecta i els resultats que es desprenen de l'estudi es presenten en el primer subapartat d'aquesta anàlisi.

Taula 1. Aplicacions considerades per a l'estudi

Nom de l'aplicació	Tipus	*Benchmark*
gcc	Enter	Spec89/92/95/2000
espresso	Enter	Spec89/92
Li	Enter	Spec89/92
fppp	Coma flotant	Spec89/95
doducd	Coma flotant	Spec89/95
tomcatv	Coma flotant	Spec89/95

A més a més, posteriorment es mostra i s'estudia l'impacte de limitar algunes de les característiques d'aquest tipus de processadors que més significació tenen en el rendiment de les aplicacions. Els resultats obtinguts, i també les característiques de la màquina que es van limitant, es mostren en els apartats posteriors.

2.1. Màquina perfecta

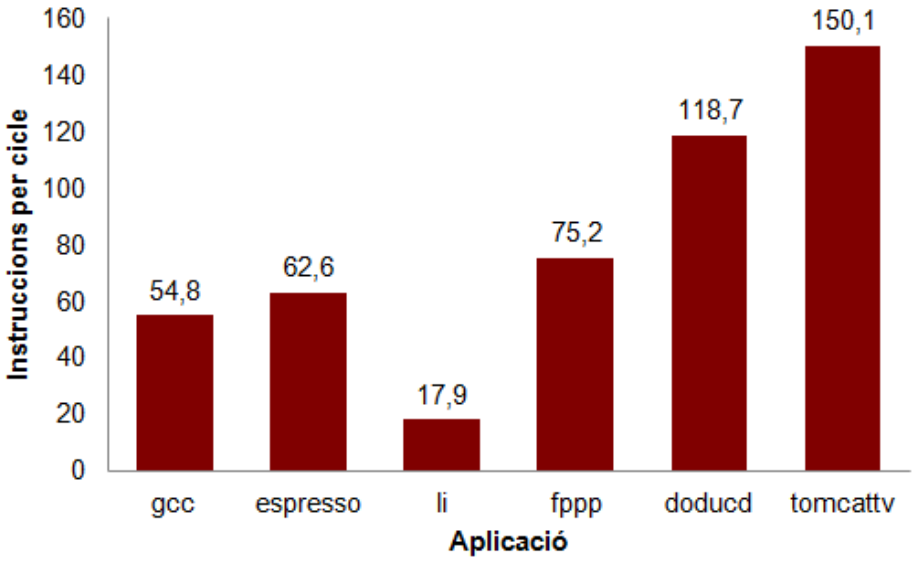
La taula 2 presenta el model de màquina ILP infinita que s'estudia. Entre altres característiques, es mostra la quantitat d'instruccions que es poden generar per cicle, el nombre d'instruccions que pot tenir en vol o el nombre de registres de reanomenament o *renaming* que tenen. D'altra banda, per tal de comparar-la amb les característiques que presenta un processador del mercat, al costat es mostren els valors que presenta una màquina Power 5 IBM (Chase i Doyle, 2001) en cadascuna de les característiques especificades.

Taula 2. Màquina ILP infinita enfront d'una Power 5

	Model	**Power 5**
Instruccions generades per cicle	Infinites	4
Finestra d'instruccions	Infinita	200
Nombre de registres per a fer reanomenament	Infinits	48 registres tipus enter 40 registres tipus coma flotant
Algorisme de predicció	Perfecte	D'un 2% a un 6% d'error
Memòria cau	Perfecta	64 kB de primer nivell de dades 64 kB de primer nivell d'instruccions 1,92 MB de segon nivell 36 MB de tercer nivell

La figura 1 mostra les instruccions per cicle que pot assolir cadascun dels *benchmarks* en aquesta màquina ideal. Com es pot observar, tant el *fpppp* com el *doducd* com el *tomcattv* mostren un nivell de paral·lelisme d'instrucció força elevat, atès que es poden fer més de setanta instruccions per cicle. El *gcc*, l'*espresso* i *li* mostren uns nivells de paral·lelisme força inferior. Sobretot tenint en compte que disposen d'un processador amb recursos infinits.

Figura 1. Rendiment en una màquina ILP ideal



A continuació, es presenten els estudis duts a terme limitant alguna o algunes de les característiques de la màquina perfecta.

2.2. Impacte de la finestra d'intruccions

Com a part de l'estudi de límits, a continuació s'avalua l'impacte en els *benchmarks* anteriors de limitar els diferents recursos que hem presentat més amunt. En primer lloc, s'analitza l'efecte de tenir una finestra d'instruccions limitada. Per a aquest estudi es consideren els valors següents: instruccions infinites, 2K, 512, 128, 32, 8 i 4 instruccions.

Cal fer notar els dos aspectes següents:

- El primer és que no es limita la quantitat d'instruccions generades per cicle: es vol veure la limitació d'aquesta arquitectura per a processar operacions assumint que en genera infinites.

- El segon és que la màquina Power 5 té una finestra d'instruccions de 200 entrades. Per tant, s'avaluen valors inferiors però també valors força superiors a aquest.

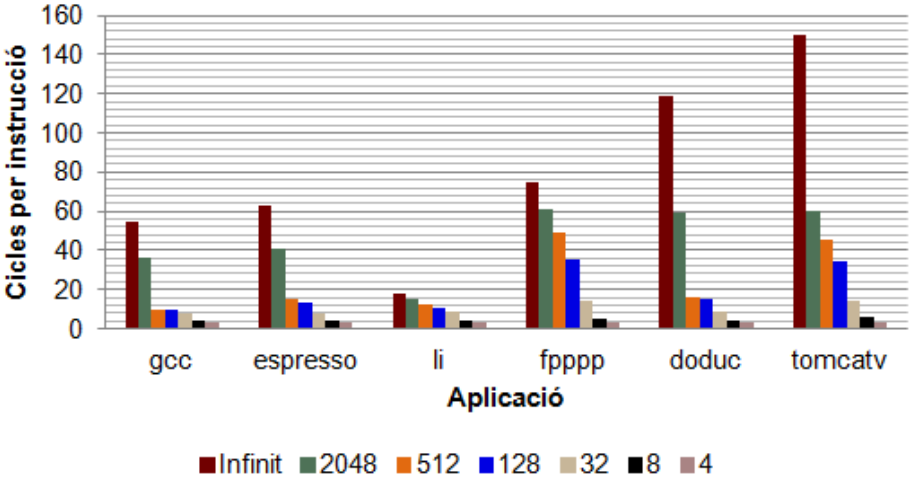
Les característiques de la màquina perfecta limitada són, doncs, les descrites en la taula 3.

Taula 3. Màquina ILP perfecta (variant de la finestra d'instrucció) enfront d'una Power 5

	Model	**Power 5**
Instruccions generades per cicle	Infinites	4
Finestra d'instruccions	Infinita, 2K, 512, 128, 32, 8, 4	200
Nombre de registres per a fer reanomenament	Infinits	48 registres tipus enter 40 registres tipus coma flotant
Algorisme de predicció	Perfecte	D'un 2% a un 6% d'error
Memòria cau	Perfecta	64 kB de primer nivell de dades 64 kB de primer nivell d'instruccions 1,92 MB de segon nivell 36 MB de tercer nivell

La gràfica següent (figura 2) mostra els efectes de limitar la finestra d'instruccions en el processador ILP perfecte introduït anteriorment. La primera de les barres mostra el valor de referència (una finestra infinita). Les altres sis barres de cada *benchmark* mostren el CPI que s'obtenen reduint la finestra d'instruccions a 2.048, 512, 128, 32, 8 i 4 instruccions.

Figura 2. Rendiment en una màquina ILP ideal variant de la finestra d'instruccions



Com es pot observar reduint la finestra a 2.048 instruccions la reducció de rendiment ja és considerable. Aquest impacte és molt dràstic per a aquelles aplicacions que presentaven un paral·lelisme d'instrucció més elevat. Per exemple, *tomcatv* ha passat d'un CPI de 155 a un IPC 60. És important fixar-se que aquesta reducció apareix reduint aquesta finestra a un valor molt elevat (2.048 instruccions). Els efectes són molt més devastadors quan es consideren els valors més baixos.

Prenent com a valor de referència una finestra d'instruccions de 200 entrades (Power 5), podem veure que les aplicacions experimentarien una reducció de rendiment molt considerable respecte de la nostra màquina ILP perfecta. Per exemple, en el cas del *gcc* es redueix el rendiment deu vegades.

2.3. Impacte del predictor de salts

El següent canvi aplicat a la màquina ILP perfecta és usar un algorisme de predicció de salts real. Com és sabut els costos associats a fer un error en la predicció d'un salt són força elevats. Aquests costos es veuen accentuats com més llarg és el *pipeline* del processador i com més instruccions en vol es poden tenir. En aquests casos, quan succeeix un error d'aquest tipus implica fer *rollback* de moltes transaccions en vol, aspecte altament costós.

La taula 4 presenta els sis escenaris considerats per aquest estudi, usant: un predictor perfecte, tres predictors reals (un de selectiu, un de basat en perfil i un d'estàtic) i cap predictor. Com a valors de referència podem veure que el predictor de salts de la Power 5 té un error mitjà situat entre un 2% i un 6%.

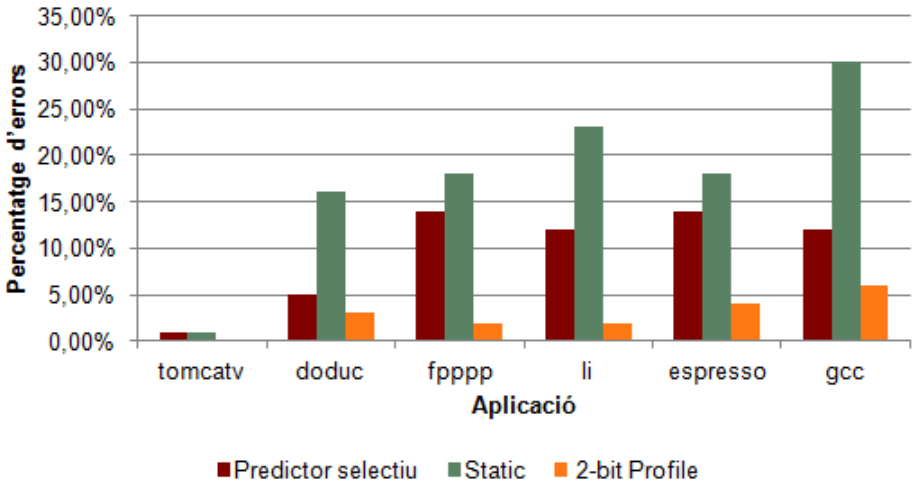
Taula 4. Màquina ILP semiperfecta (variant de l'algorisme de predicció de salts) enfront d'una Power 5

	Model	Power 5
Instruccions generades per cicle	64	4
Finestra d'instruccions	2.048	200

	Model	Power 5
Nombre de registres per a fer reanomenament	Infinits	48 registres tipus enter 40 registres tipus coma flotant
Algorisme de predicció	Perfecte, predictor selectiu, 2 bits, estàtic enfront de cap	D'un 2% a un 6% d'error
Memòria cau	Perfecta	64 kB de primer nivell de dades 64 kB de primer nivell d'instruccions 1,92 MB de segon nivell 36 MB de tercer nivell

Com es mostra en la figura 3, el percentatge d'errors de predicció que cadascun dels *benchmarks* usats en aquest estudi assoleix per a cadascun dels tres predictors emprats. Com podem observar el que té millor rendiment és el que usa dos bits per a intentar capturar el perfil del flux executat.

Figura 3. Percentatge d'errors en la predicció per aplicació variant d'algorisme de predicció de salts

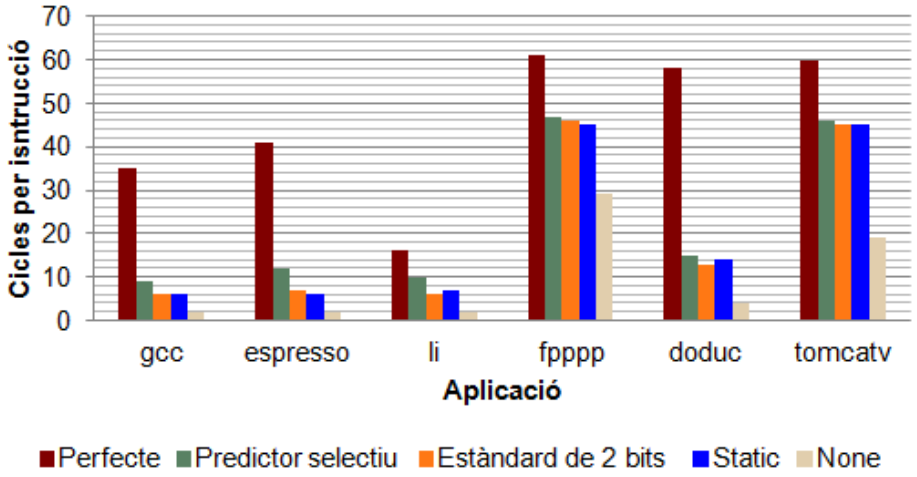


A continuació, en la figura 4 es mostra l'impacte en les instruccions per cicle (IPC) en cadascun dels *benchmarks* considerats. Com podem veure, exceptuant el *fppp* i el *tomcatv*, l'impacte és considerable. Per exemple, el *gcc* passa d'unes IPC de 35 usant un predictor perfecte a unes IPC de 5 usant un predictor basat en perfils de dos bits. És important fer notar que aquest darrer predictor mostrava un error d'un 5% en la predicció del salt. Aquests resultats demostren que l'impacte dels predictors de salt en aquest tipus d'arquitectures és molt elevat en la majoria de casos, ja que redueix fins a sis vegades el seu nombre d'instruccions per cicle retirades.

Cal destacar que el model que s'està considerant té una jerarquia de memòria perfecta. Si es considerés un model de memòria real, els errors de memòria serien molt més costosos. Els accessos a memòria són extremament costosos, per tant, els casos en què hi hagués errors en les memòries cau locals i les operacions s'haguessin de revertir a causa d'un error de predicció, s'estaria

perdent rendiment d'una manera considerable. Per tant, les IPC mostrades a continuació podrien ser substancialment inferiors si es considerés un model més proper a la realitat.

Figura 4. Rendiment en una màquina ILP semiperfecta variant de l'algorisme de predicció de salts



2.4. Impacte dels registres de reanomenament o *renaming*

En els darrers subapartats s'ha mostrat l'impacte de variar la finestra d'instruccions i l'algorisme de predicció de salts en un processador en què només hi ha un flux d'execució. Com s'ha esmentat, el rendiment és marcat per les no-dependències entre les instruccions que s'executen. Una de les tècniques més emprades en aquests tipus d'arquitectura és el reanomenament (*renaming*) de registres.

A continuació, en la taula 5 es mostra una nova extensió de la màquina ILP anterior variant el nombre de registres de reanomenament (tant de coma flotant com enters). Com es pot observar, en aquest cas s'ha fixat l'algorisme de predicció de salts al perfil de 2 bits. La Power 5 té un total de 48 i 40 registres de reanomenament, de tipus enter i de coma flotant, respectivament. El model que estudiarem en aquest apartat té registres de reanomenament infinits, 256, 128, 64 i 32 registres tant de coma flotant com enters.

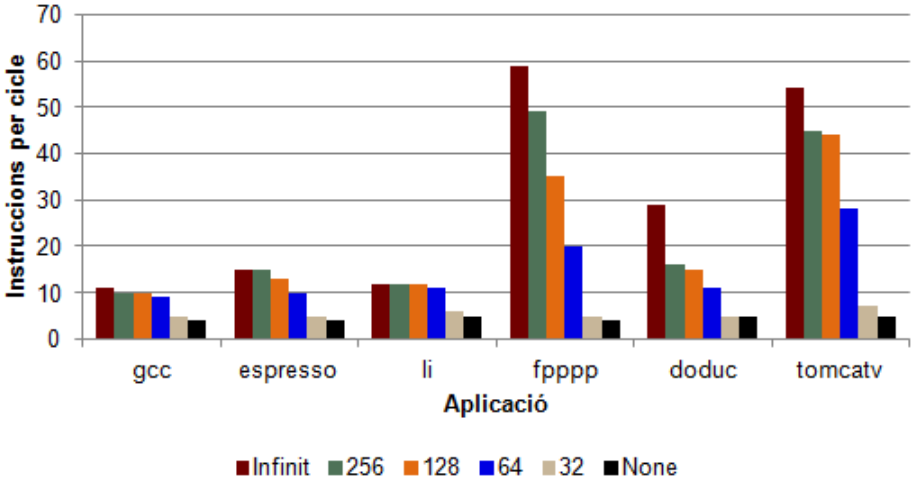
Taula 5. Màquina ILP semiperfecta (variant del nombre de registres de reanomenament) enfront d'una Power 5

	Model	Power 5
Instruccions generades per cicle	64	4
Finestra d'instruccions	2.048	200
Nombre de registres per a fer reanomenament	Infinits, 256, 128, 64, 32, cap (tant enter com coma flotant).	48 registres tipus enter 40 registres tipus coma flotant
Algorisme de predicció	Perfil 2 bits	D'un 2% a un 6% d'error

	Model	Power 5
Memòria cau	Perfecta	64 kB de primer nivell de dades 64 kB de primer nivell d'instruccions 1,92 MB de segon nivell 36 MB de tercer nivell

La figura 5 mostra la variació en les IPC per a cadascun dels *benchmarks* considerats i les diferents quantitats de registres de reanomenament. És interessant notar l'impacte d'aquesta variable en les tres aplicacions que mostraven un índex de paral·lelisme a nivell d'aplicació més alt. Tant el *fppp*, com el *doduc*, com el *tomcat* mostren una degradació important en reduir la quantitat de registres de reanomenament. Si s'agafa com a referència el de 64 registres (proper a la quantitat de registres que té la Power 5), es pot observar que les IPC es redueixen un 50% respecte d'una mida infinita. Si es redueix a 32 entrades, les IPC ja es redueixen notòriament. En aquest cas, el rendiment cau fins a deu vegades respecte d'una mida infinita.

Figura 5. Rendiment en una màquina ILP semiperfecta variant de la quantitat de registres de reanomenament



2.5. Conclusions

Durant aquest apartat s'ha vist l'impacte de passar d'una màquina amb una arquitectura perfecta amb recursos infinits a una màquina acotada però amb recursos considerablement alts. Cal fer notar que tenir més recursos no és gratuït, és a dir, tenir molts registres de reanomenament o una finestra d'instrucció molt gran implica que l'àrea que requereix el processador incrementa notablement i el consum d'energia es dispara. Això es contraposa a les tendències actuals, en què s'acostuma a intentar reduir el consum i treure rendiment del paral·lelisme a nivell de fil.

Així, doncs, el model resultant de tots els canvis (finestra d'instruccions de 2.048, predictor de perfil de 2 bits, generació de 64 instruccions per cicle, memòria cau infinita) ha mostrat que es passa ràpidament d'unes IPC molt elevades de la màquina infinita a unes IPC molt més moderades en una màqui-

na limitada més propera a la realitat. Cal destacar que, tot i ser una màquina ILP limitada, els valors emprats per aquests estudis han estat molt elevats. Per exemple, s'ha assumit que podia generar 64 instruccions per cicle, tenia un banc de registre de 256 elements (tant coma flotant com enters) i tenia una finestra d'instrucció de 2.048 instruccions. De la màquina infinita i perfecta original s'han reduït les IPC substancialment en tots els *benchmarks* considerats (de cinc a deu vegades).

3. Paral·lelisme a nivell de fil d'execució o *multithreading*

En l'apartat anterior s'han estudiat els límits arquitectònics dels processadors que intentaven treure rendiment amb el paral·lelisme a nivell d'instrucció. No obstant això, el rendiment de les aplicacions no escala proporcionalment a la quantitat de recursos afegits. Fins i tot, en molts casos, per més recursos que s'hi afegeixin, el rendiment de l'aplicació es veu incrementat només lleugerament.

Aquest factor és el que va motivar l'aparició d'arquitectures que consideren l'execució simultània de diferents fils d'execució en un mateix processador: paral·lelisme a nivell de fil o *thread level parallelism* (TLP) (Lo, 1997). En aquestes arquitectures:

- Diferents fils d'execució comparteixen les unitats funcionals del processador (per exemple, unitats funcionals).

- El processador ha de tenir estructures independents per a cadascun dels fils que executa: registre de reanomenament, comptador de programa, etc.

- Si els fils pertanyen a diferents processos, el processador ha de facilitar mecanismes perquè puguin treballar amb diferents taules de pàgines.

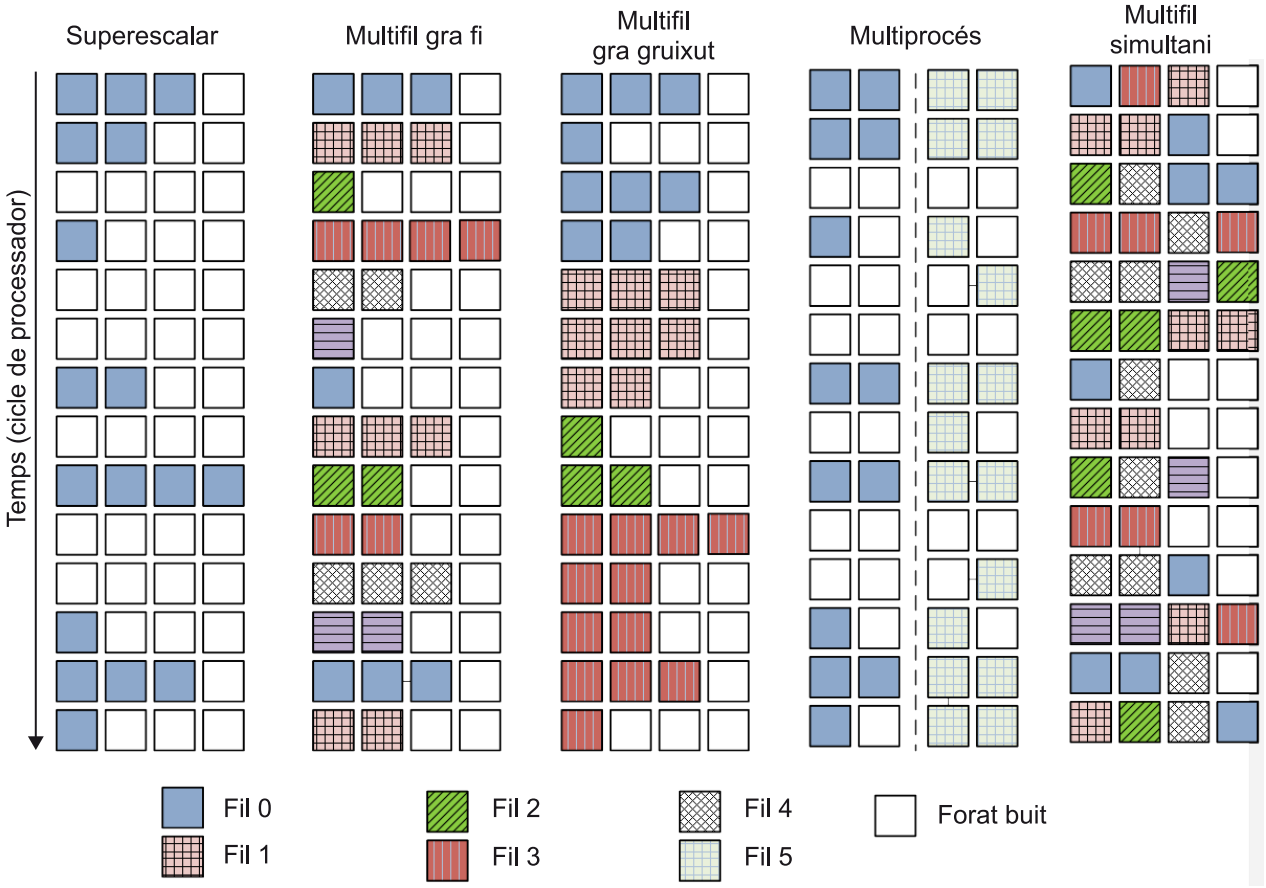
Com es mostra durant els apartats següents les arquitectures actuals exploten aquest paradigma de moltes maneres diferents. No obstant això, la motivació és la mateixa: la majoria d'aplicacions actuals tenen un alt nivell de paral·lelisme i el seu rendiment escala afegint més paral·lelisme a nivell de processador. L'objectiu és millorar la productivitat dels sistemes que poden córrer aplicacions que són inherentment paral·leles. Exemples d'aquest tipus d'aplicacions són: navegadors, bases de dades, servidors web, etc.

En aquests entorns actuals, treure rendiment explotant el TLP pot ser molt més efectiu en termes de cost/rendiment que explotant l'ILP. En la majoria de casos, com més paral·lelisme es faciliti, més rendiment se'n podrà treure. En qualsevol cas, tal com s'estudia més endavant, la majoria de tècniques que s'havien emprat per a sistemes ILP també són emprades en sistemes TLP.

A continuació, es presenten els diferents tipus d'arquitectures TLP que s'han proposat durant les darreres dècades. La figura 6 mostra com s'executarien diferents fils d'execució en cadascuna de les arquitectures introduïdes a continuació.

a) Arquitectures multiprocessador (MP). Aquesta és l'extensió més senzilla a un model ILP. En aquest cas repliquem una arquitectura ILP *n* vegades. El model més bàsic d'aquestes arquitectures és el multiprocessador simètric. El desavantatge principal és que, tot i tenir molts fils d'execució, cadascun d'aquests fils continua tenint les limitacions d'un ILP. No obstant això, com es mostra més endavant hi ha variants en les quals cada processador és alhora multifil.

Figura 6. Model ILP enfront de models TLP



b) Arquitectures multifil, també conegudes com a ***super-threading***. Aquesta va ser la següent de les variants TLP que va aparèixer. El model de *pipeline* del processador s'estén considerant també el concepte de *fil d'execució*. En aquest cas, el planificador (que escull quina de les instruccions començarà en aquest cicle) té la possibilitat d'escollir quin dels fils d'execució començarà la instrucció següent en el cicle següent. Per exemple, TERA Systems (Alverson, Callahan, Cummings i Koblenz, 1990) podia treballar amb 128 fils a la vegada.

c) Arquitectures amb execució de fil simultània o *simulatenous multithreading* (SMT). Aquestes són una variació de les arquitectures multifil que permet a la lògica de planificació escollir instruccions de qualsevol fil a cada cicle de rellotge. Aquesta condició fa que la utilització dels recursos sigui molt més elevada i eficient. El desavantatge més gran d'aquest tipus d'arquitectures és la complexitat de la lògica necessària per a dur a terme aquesta gestió. El fet de poder començar diverses instruccions de diferents fils és molt costós. Per aquest motiu, el nombre de fils que aquestes arquitectures acostumen a fer

servir és relativament baix. Per exemple, les arquitectures Intel amb *hyperthreading* (Marr, 2002) implementen dos o més fils d'execució, o bé l'Alpha 21464 (Seznec, Felix, Krishnan i Sazeide, 2002) implementa quatre fils d'execució.

d) Arquitectures *multicore*, anomenades *chip-multiprocessor* (CMP) o *system-on-chip* (SOC). Conceptualment són similars a les primeres arquitectures esmentades, però a escala més petita. En el cas de sistemes MP, hi ha *n* processadors independents que poden compartir la memòria però no comparteixen recursos entre ells, com ara una memòria cau de darrer nivell. Els SOC són una evolució conceptual dels MP, però traslladada a nivell de processador. En un SOC un mateix processador es compon per *m* nuclis en què poden córrer fils que poden estar relacionats o fins i tot poden compartir recursos.

Durant els propers apartats s'estudien cadascuna d'aquestes arquitectures.

4. Arquitectures *super-threading*

El TLP treu rendiment pel fet de compartir els recursos entre diferents fils d'execució. No obstant això, hi ha dues maneres de dur a terme aquesta compartició. La primera consisteix a compartir els recursos en l'espai i el temps, és a dir, en un cicle concret i en una etapa concreta del processador, instruccions de fils diferents poden estar compartint les mateixes etapes del processador. La segona consisteix a compartir els recursos en el temps, és a dir, en un cicle concret i en una etapa concreta del processador, només s'hi poden trobar instruccions d'un mateix fil. Aquest segon tipus de compartició és coneguda com a *super-threading*.

Dins les arquitectures *super-threading*, hi ha dues maneres de compartir els recursos en el temps entre els diferents fils. Les més habituals són: compartició fina o compartició gruixuda.

4.1. Compartició fina

En la compartició de recursos fina, el processador canvia de fil a cada instrucció que aquest executa. D'aquesta manera, l'execució dels fils es fa de manera intercalada. Habitualment, el canvi de fil es fa seguint una distribució *round robin*, és a dir, s'executa una instrucció del fil n, després del $n + 1$, del $n + 2$, etc. En els casos en què un fil es troba bloquejat, per exemple esperant dades de memòria, se salta i es passa al següent. Per tal de poder dur a terme aquest tipus de canvis el processador ha ser capaç de fer un canvi de fil per cicle.

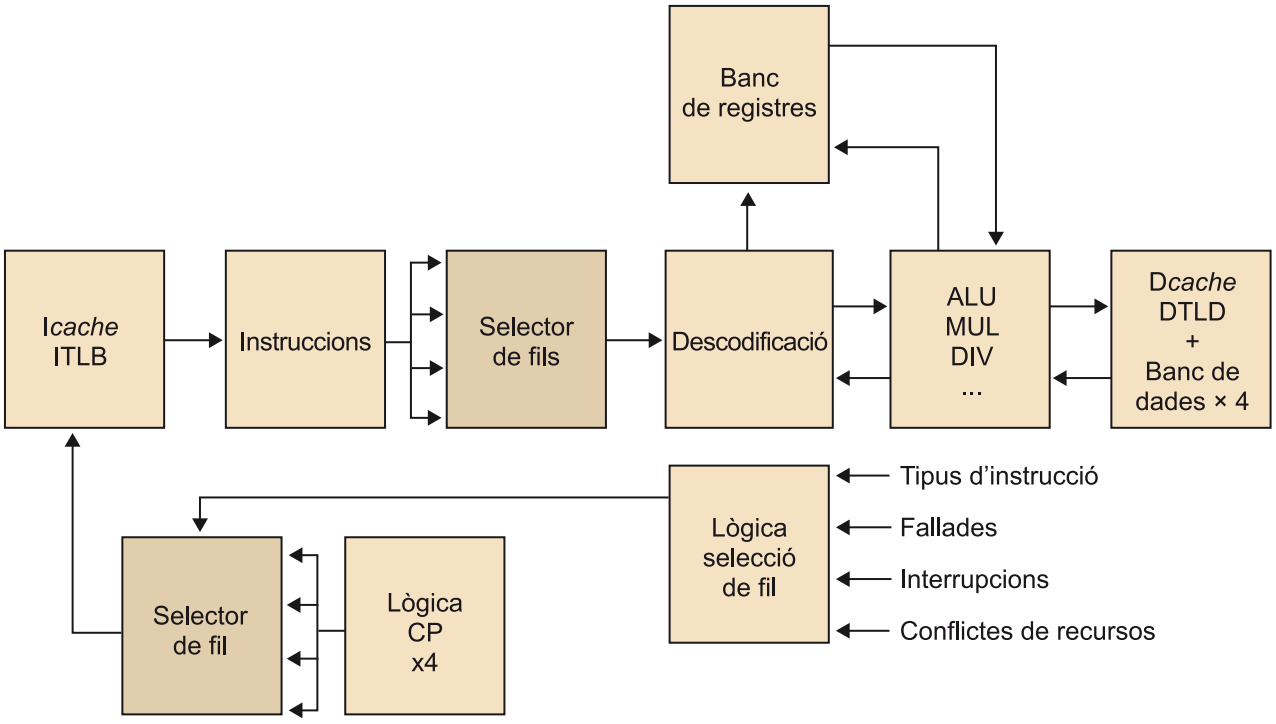
L'avantatge d'aquesta opció és que pot esmorteir bloquejos curts i llargs dels fils que s'estan executant, atès que a cada cicle es canvia de fil. El desavantatge principal és que es redueix el rendiment dels fils de manera individual, ja que fils preparats per a executar instruccions es veuran atraçats per les instruccions dels altres fils. Això té implicacions de complexitat de disseny i de consum d'energia, atès que el processador ha de poder canviar de fil. Un exemple d'aquest tipus de processador és l'UltraSPARCT1 (Kongetira, Aingaran i Olukotun, 2005).

El processador UltraSPARC T1, també conegut com a *Niagara*, va ser anunciat per l'empresa SunMicrosystems el novembre del 2005. Aquest nou processador UltraSPARC era multifil i multinucli, dissenyat per arquitectures de tipus servidor amb un consum energètic baix. Per exemple, a 1,4 GHz consumeix aproximadament uns 72 W. Anteriorment, l'empresa Sun ja havia introduït dues arquitectures multinucli: els UltraSPARC IV i IV+. No obstant això, T1 va ser el primer microprocessador que era multinucli i multifil. Se'n poden trobar

versions amb quatre, sis o vuit nuclis, cadascun dels quals pot manejar quatre fils concurrentment. Això implica que es poden córrer fins a un màxim de 32 fils concurrentment.

La figura 7 mostra les etapes (també anomenat *pipeline*) que cadascun d'aquests nuclis té. Com es pot observar conté un selector que decideix quin fil s'executa en cada cicle. Aquest selector va canviant de fil a cada cicle. En qualsevol cas, només escull sobre el conjunt de fils disponibles en cada moment. Quan succeeix un esdeveniment de llarga latència (com, per exemple, un error de memòria cau que desencadena una petició a memòria), el fil que l'ha causat es treu de la cadena de rotació (*round robin*). Un cop aquest esdeveniment de llarga durada acaba, el fil es torna a afegir a la rotació per a accedir a les unitats funcionals. El fet que el *pipeline* es comparteixi amb diferents fils cada cicle fa que cada fil vagi més lent, però la utilització del processador és més elevada. Un altre efecte molt interessant és que l'impacte d'error de les memòries cau és molt més reduït: en cas que un dels fils causi un error, els altres poden continuar usant els recursos i progressant.

Figura 7. *Pipeline* UltraSPARC T1



Com es pot observar en la figura anterior, considerant un cicle determinat, tots els elements del *pipeline* són usats per només un fil. No obstant això, algunes de les estructures es troben replicades pel nombre de fils que el nucli té, com, per exemple, la lògica de gestió del comptador de programa. En aquest cas, el nucli ha de ser capaç de distingir en quina part del flux es troba cada fil. Per tant, necessita tenir aquesta estructura per a cadascun dels fils. Contràriament, la resta de lògica, com, per exemple, la lògica de descodificació, és única i només usada per un sol fil en un cicle determinat.

4.2. Compartició gruixuda

En la compartició gruixuda els canvis de fils es fan quan en el fil que està corrent succeeix un bloqueig de llarga durada. Un exemple d'aquest tipus de bloqueig és un error en la memòria cau de darrer nivell. En aquest cas, caldria anar a la memòria, fet que implicaria molts cicles de bloqueig.

L'avantatge d'aquesta opció és que no es redueix el rendiment del fil individual, perquè només canvia de fil quan aquest és bloquejat per un esdeveniment que requereix una llarga latència per a ser processat. El desavantatge és que no és capaç de treure'n rendiment quan els bloquejos són més curts. Com que el nucli només genera instruccions d'un sol fil en cada cicle, quan aquest es bloqueja en un cicle concret (per exemple, per una dependència entre instruccions), totes les etapes següents del processador queden bloquejades o congelades. Aquestes es tornaran a emprar tan bon punt el fil pugui tornar a processar instruccions.

Un altre desavantatge important és que els nous fils que comencen al processador han de passar per totes les etapes abans que el processador comenci a retirar instruccions per aquest fil. En el model anterior, com que cada cicle es canvia de fil el rendiment la productivitat no es veu tan afectada. Per exemple, suposem un processador amb dotze cicles d'etapes i quatre fils d'execució. En el millor dels casos, un nou fil tardarà dotze cicles a retirar la primera instrucció. Però durant aquests dotze cicles, en el millor dels casos, els altres tres fils hauran pogut retirar dotze instruccions. En canvi, en el gra gruixut hauríem estat dotze cicles sense retirar-ne cap.

Un exemple d'aquest tipus de processador és l'IBM AS/400 (IBM, 2011).

5. Arquitectures amb multifil simultani

Les arquitectures amb multifil simultani, *simultaneous multithreading* (SMT) (Tullsen, Eggers i Levy, 1995), són una variació de les arquitectures tradicionals multifil. Aquestes noves arquitectures permeten escollir instruccions de qualsevol dels fils que el processador està executant en cada cicle de rellotge. Això vol dir que diferents fils estan compartint en un mateix instant de temps diferents recursos del processador. Aquesta compartició és tant horitzontal (per exemple, etapes successives del *pipeline*) com vertical (per exemple, recursos d'una mateixa etapa del processador).

El resultat d'aquestes tècniques és una alta utilització dels recursos del processador i molta més eficiència. Cal recordar que, a diferència de les arquitectures paral·leles anteriors, en un mateix instant de temps, dos fils poden estar compartint els mateixos recursos d'una de les etapes del processador. No obstant això, aquesta eficiència no és gratuïta, és a dir, per a donar suport a aquesta compartició el processador conté una lògica extra i força complexa. Cal fer notar que, per defecte, els fils de diferents processos no s'han de poder veure entre ells, tant per qüestions de seguretat com de funcionalitat, l'execució d'una instrucció A d'un fil X no pot modificar el comportament funcional d'un fil B. Cal recordar que en un sistema operatiu cada procés té el seu propi context i que aquest és independent dels contextos dels altres processos, sempre que no s'usin tècniques explícites per a compartir recursos.

Les arquitectures que són totalment SMT i que són capaces de treballar amb molts fils són extremament costoses en termes de complexitat. És important tenir en compte que les estructures necessàries per tal de suportar aquest tipus de compartició creix proporcionalment al nombre de fils disponibles. Per tant, si bé és cert que amb més paral·lelisme s'obté més rendiment, com més paral·lelisme, més àrea i més consum energètic. En aquests casos, cal aconseguir un balanç de rendiment davant de consum energètic, complexitat i àrea. Per exemple, l'Hyper-Threading d'Intel implementa només contextos i dos fils. Un altre exemple és l'Alpha 21464 que disposa de fins a quatre fils paral·lels.

En la resta de l'apartat es discuteix el disseny d'aquest tipus d'arquitectures, i també algunes de les arquitectures SMT més rellevants de la bibliografia.

5.1. Conversió del paral·lelisme a nivell de fil a paral·lelisme a nivell d'instrucció

Com s'ha comentat, les arquitectures SMT permeten la compartició de diferents unitats funcionals per diferents fils d'execució. Per a permetre aquest tipus de compartició, l'estat de cadascun dels fils ha de ser guardat de manera independent. Per exemple, cal tenir per duplicat els registres que els fils fan

servir, el seu comptador de programa i també una taula de pàgines separada per fil. En el cas de no tenir aquestes estructures duplicades, l'execució funcional de cadascun dels fils interferiria amb la d'un altre fil. D'altra banda, hi podria haver problemes de seguretat greus. Per exemple, compartir la taula de pàgines implicaria que un fil compartiria el mapatge d'adreces virtuals a físiques. Això implicaria que un fil podria accedir a la memòria de l'altre fil sense cap tipus de restricció. No obstant això, hi ha altres recursos que no cal replicar, com, per exemple, l'accés a les unitats funcionals per a accedir a memòria (atès que els mecanismes de memòria virtual ja hi donen suport per multiprogramació).

La quantitat d'instruccions que un processador SMT pot generar per cicle es troba limitada pels desbalancejos en els recursos necessaris per a executar els fils i la disponibilitat d'aquests recursos. No obstant això, també hi ha altres factors que limiten aquesta quantitat. Per exemple, el nombre de fils actius, possibles limitacions en la mida de les cues disponibles, la capacitat de generar prou instruccions dels fils disponibles o limitacions de quin tipus d'instruccions es poden generar des de cada fil i quines poden generar tots els fils.

Les tècniques SMT assumeixen que el processador facilita un conjunt de mecanismes que permeten l'explotació del paral·lelisme a nivell de fil. En particular, aquestes arquitectures tenen un conjunt gran de registres virtuals que poden ser usats per a guardar els registres de cadascun dels fils de manera independent (assumint, és clar, diferents taules de reanomenament per a cada fil). El reanomenament de registres facilita identificadors únics de registre; sense aquest tipus de reanomenament dos fils podrien tenir interferències entre les seves execucions.

Per exemple, el flux d'execució dels dos fils que es presenten a continuació no funcionarien correctament (taula 6). Si els registres r2, r3 i r4 no fossin reanomenats per a cadascun dels fils, l'execució funcional dels fils seria errònia. En els instants 1 i 4 els valors que ambdós fils llegirien serien incorrectes. En el primer cas, el fil 2 estaria emprant el valor del registre r3 modificat pel fil 1 en el cicle anterior. De manera similar, també succeiria en el cicle 3, en què el fil 1 estaria sumant un valor r3 modificat per un altre fil.

Taula 6. Exemple de flux d'instruccions multifil

cicle (n)	fil 1	-->	"LOAD #43, r3"
cicle (n+1)	fil 2	-->	"ADD r2, r3, r4"
cicle (n+2)	fil 1	-->	"ADD r4, r3, r2"
cicle (n+3)	fil 1	-->	"LOAD (r2), r4"

Gràcies al reanomenament de registres les instruccions de diferents fils poden ser barrejades durant les diferents etapes del processador sense confondre fonts i destinacions entre els diferents fils disponibles. És important fer notar que aquest tipus de tècnica és la que s'empraria en els processadors fora d'ordre, en què es té una taula de reanomenament per fil d'execució. No obstant això,

en aquest cas també hi ha una lògica diferent per tal de guardar els diferents comptadors de programa (un per fil), hi ha la capacitat de finalitzar més d'una instrucció per cicle, es tenen predictors de salts per fil, etc.

Així, doncs, el procés de reanomenament de registres és exactament el mateix procés que fa un processador fora d'ordre. Per aquest motiu, un SMT es pot considerar una extensió d'aquest tipus de processadors (afegint, és clar, tota la lògica necessària per a suportar els contextos dels diferents fils). Com es pot deduir, es pot construir una arquitectura fora d'ordre i SMT a la vegada.

El procés de finalització d'una instrucció (*commit*, en anglès) no és tan senzill com en un processador no SMT (en què la finalització només es té en compte un fil). En aquest cas, es vol que la finalització d'una instrucció sigui independent per a cada fil. D'aquesta manera, cadascun pot avançar de manera independent als altres. Això es pot dur a terme amb les estructures que permeten aquest procés per a cadascun dels fils (per exemple, tenint un *reoder buffer* per fil).

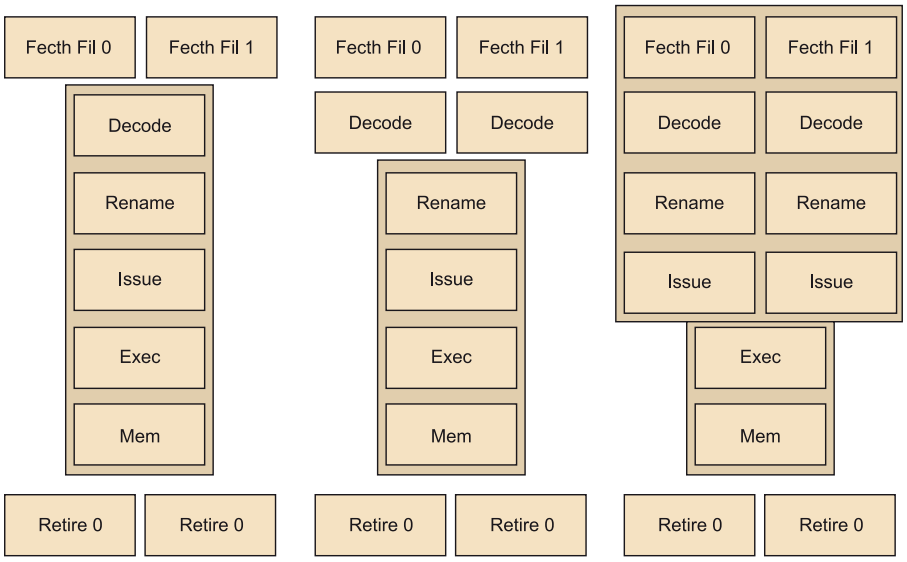
5.2. Disseny d'un SMT

En termes generals, els SMT segueixen la mateixa arquitectura que els processadors superescalars. Això inclou tant els dissenys generals de les etapes que els componen (etapa de cerca d'instrucció, etapa de descodificació i lectura de registres, etc.), com les tècniques o els algorismes emprats (per exemple, Tomasulo).

No obstant això, tal com s'ha discutit en l'apartat anterior moltes de les estructures han de ser replicades per a donar suport als diferents contexts que el processador ha de gestionar. Algunes són les mínimes necessàries per tal d'evitar interferències entre fils i assegurar la correctesa de les aplicacions (per exemple, tenir comptadors de programes separats o taules de pàgines separades). Tanmateix, es poden trobar variants arquitectòniques que no són estrictament necessàries, però que poden donar més rendiment en certes situacions.

La figura 8 mostra algunes de les opcions que es poden tenir en compte quan es considera l'arquitectura global d'un processador SMT. En aquesta figura es mostren diferents possibilitats de com els diferents fils comparteixen o no les diferents etapes del processador (s'han assumit les etapes típiques d'un processador superescalar fora d'ordre). La primera de totes assumeix que tan sols la cerca d'instrucció es troba dividida per fil, la resta d'etapes són compartides. Com més a la dreta ens movem en la figura, les etapes es troben més dividides per fil. Cal remarcar que el fet que una etapa es trobi separada per fils exemplifica que el processador fa l'etapa dividida per fils i que cada fil té estructures separades per a dur-la a terme. No obstant això, és lògic pensar que tots els fils tindran un lògica compartida atès que el mecanisme és comú entre tots ells.

Figura 8. Possibles disseny d'una arquitectura SMT



En tots els casos, les etapes d'execució i accés a memòria són compartides per tots els fils. En un estudi acadèmic es podria considerar que es troben replicades per fil, però, ara com ara, en un entorn real, això és molt costós en termes d'espai i de consum energètic. D'altra banda, com és lògic, la utilització d'aquests recursos serà molt més elevada en els casos en què tots els fils els comparteixin. Així, doncs, quan els uns estiguin bloquejats els altres l'usaran i viceversa, o bé quan els uns estigui fent accessos a memòria, els altres podran usar les unitats aritmeticològiques. Per tant, per tal de maximitzar l'eficiència energètica del processador és necessari que siguin compartides.

En aquests exemples l'etapa de cerca d'instrucció se separa per fil. Tanmateix, això no és comú a tots els dissenys possibles. Com es veurà més endavant l'Hyper-Threading d'Intel conté una etapa de cerca d'instrucció compartida per tots els fils. En la cerca s'inclou la lògica de selecció de fil, i també el predictor de salts. També cal considerar que l'accés a la memòria cau d'instruccions és independent per fil. Aquesta memòria ha de tenir ports de lectura suficients per tal de satisfer la necessitat d'instruccions dels fils.

Les etapes de descodificació i reanomenament identifiquen les fonts i destinacions de les operacions, i també calculen les dependències entre les instruccions en vol. En aquest cas, com que les instruccions dels diferents fils seran independents, podrien tenir la lògica corresponent de manera separada. No obstant això, tenir les estructures de reanomenament compartides fa que el processador pugui ser més eficient en la majoria de casos. Per exemple, si s'assumeix que es disposa d'un total de 50 registres de reanomenament per a ambdós fils i les estructures estan separades, cada fil podrà tenir accés a 25 registres com a màxim. En aquells casos en què un del fils només en pugui emprar 10, però l'altre en necessiti 40, el sistema estarà infrautilitzat, de manera que el rendiment del segon fil es veurà reduït substancialment.

5.3. Complexitats i reptes en les arquitectures SMT

Les arquitectures SMT incrementen notòriament el rendiment del sistema augmentant la finestra d'instruccions que pot gestionar. D'aquesta manera, en un mateix cicle el processador pot escollir un ventall ampli d'instruccions dels diferents fils disponibles al sistema. Tanmateix, la resta d'etapes es troben també més utilitzades pel mateix motiu. No obstant això, cal tenir en compte que aquestes millores ocorren en contra del rendiment individual del fil. En aquest cas, el rendiment que un sol fil pot aconseguir pot ser inferior al que hauria obtingut en un processador superescalar fora d'ordre sense SMT.

Per a evitar aquest detriment individual en els fils, alguns d'aquests processadors introdueixen el concepte de *fil preferit* (*preferred thread*). La unitat encarregada de generar o començar instruccions donarà preferència als fils preferits. *A priori* pot semblar que aquest aspecte pot afavorir el fet que alguns dels fils tinguin un rendiment més alt i que no se sacrifiqui el rendiment global del sistema.

Ara bé, això no és del tot cert, perquè donant preferència a un subconjunt de fils es provoca una disminució en l'ILP del flux d'instruccions que circulen pel *pipeline* del processador. El rendiment de les arquitectures SMT es maximitza quan hi ha un nombre suficient de fils independents que permetin esmorteir els bloquejos que cadascun experimenta durant la seva execució.

Cal comentar que també hi ha algunes arquitectures en què només es consideren els fils preferits, sempre que aquests no es bloquegin. En el moment en què un d'aquests no pot seguir endavant, el processador considera els altres fils. En aquest cas, el que s'està fent és causar un desbalanceig de rendiment als diferents fils que el processador executa. Aquest factor s'ha de tenir en compte alhora de planificar l'execució dels diferents fils que corren sobre el sistema operatiu.

A banda del repte que les arquitectures SMT mostren vers la millora del rendiment individual dels fils, hi ha una varietat d'altres reptes que cal afrontar en el disseny d'un processador d'aquestes característiques. Per exemple:

- Mantenir una lògica simple en les etapes que són fonamentals i que cal executar en un sol cicle. Per exemple, en la tria d'instrucció simple cal tenir present que com més fils hi ha, la bossa d'instruccions que es poden escollir és més gran. El mateix succeeix en l'etapa de finalització d'instrucció, en què el processador ha d'escollir quines de les instruccions acabades finalitzaran en el proper cicle.

- Tenir diferents fils d'execució implica que cal tenir un banc de registres prou gran per tal de guardar cadascun dels contextos. Això té implicacions tant d'espai com de consum energètic.

- Un dels problemes de tenir diferents fils d'execució compartint els recursos d'un mateix processador pot ser l'accés compartit a la memòria cau. Hi pot haver circumstàncies en què els mateixos fils estiguin fent el que s'anomena *compartició falsa*. Aquesta succeeix quan fils diferents estan compartint els mateixos *sets* de la memòria cau, tot i que estan accedint a adreces físiques diferents. En els casos en què hi hagi molta compartició falsa el rendiment del sistema es degradarà de manera substancial.

Durant el apartats següents es presentaran dues tecnologies comercials que van incorporar el concepte de *multifil compartit* en els seus dissenys: l'Hyper-Threading d'Intel i el processador 21464 d'Alpha.

Altres exemples de processadors van ser: MemoryLogix MLX1 (Song, 2002), ClearwaterNetwroks CNP810SP (Melvin, 2000) o FlowStromPorthos (Melvin, Flowstorm porthos massive multithreading (mmt) packet processor, 2003).

5.4. Implementacions comercials de l'SMT

5.4.1. L'Hyper-Threading d'Intel

Hyper-Threading va ser el nom comercial amb el qual l'empresa Intel va introduir al mercat les tecnologies SMT en els seus productes. La primera família de productes en incloure *hyperthreading* va ser la gamma de processadors Xeon orientada a servidors, llançada el 2002. El novembre d'aquest any Intel va llançar el processador Intel Pentium 4. Aquest va ser el primer a incloure SMT en el seu disseny.

El 2010, Intel va llançar el processador Atom. Aquest també inclou la tecnologia SMT. No obstant això, aquest producte està orientat al mercat dels dispositius de baix consum. Això inclou portàtils de baix consum, tauletes, telèfons mòbils, etc. Per aquest motiu, tot i ser una tecnologia SMT, no és fora d'ordre. Per tant, no inclou tècniques de reordenament d'instruccions, execució especulativa o reanomenament de registres.

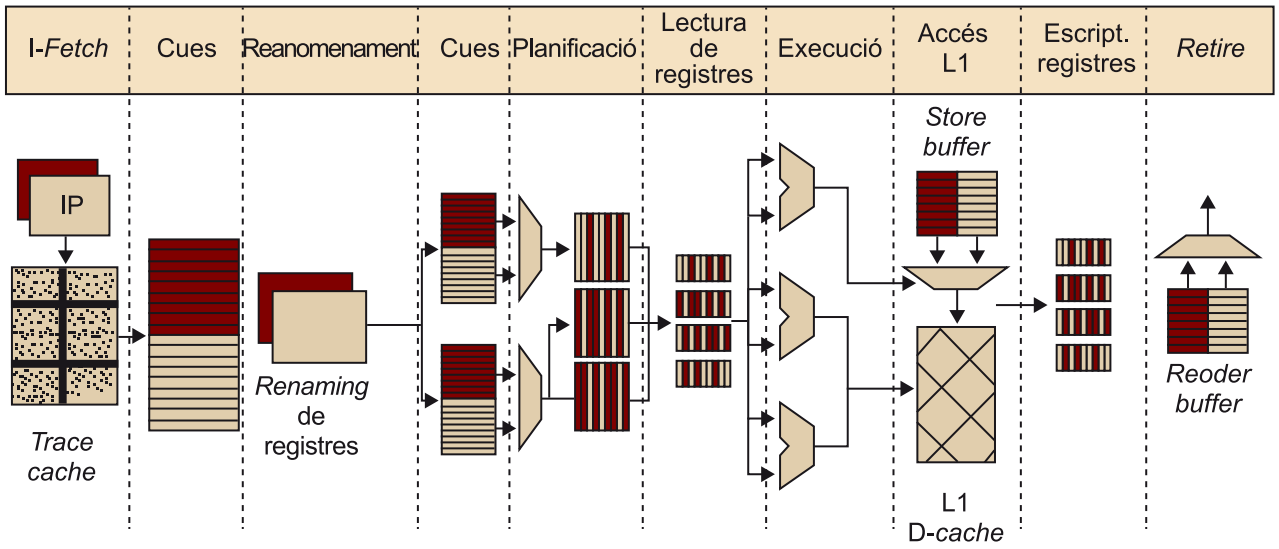
Dins de la gamma dels productes Xeon (Intel), i de manera més recent, Intel va reintroduir el concepte d'*hyperthreading* en la gamma de processadors Nehalem. En aquest cas, com que el segment de mercat destinat són els servidors, els processadors ofereixen un nivell de paral·lelisme molt més elevat i totes les funcionalitats d'un processador fora d'ordre. En aquest cas, ens movem del processador Bloomfield, amb quatre nuclis i un total de vuit fils d'execució, fins al processador Beckton, amb vuit nuclis i un total de setze fils d'execució. En aquests casos cal fer notar que la tecnologia SMT s'inclou dins de cada nucli. Més endavant, es presentaran les arquitectures multinucli.

Com es pot observar, la tecnologia SMT és aplicable a molts tipus de segments: segment de dispositius mòbils o segments de computació d'altes prestacions. El concepte de base en tots els casos és el mateix: potenciar el rendiment del sistemes augmentant el nivell de paral·lelisme de les aplicacions. En uns casos es necessita un rendiment molt elevat (per exemple, servidor de bases de dades) i en d'altres, només poder tenir diferents fils que s'executin en paral·lel (per exemple, el navegador i el gestor de correu). El consum i complexitat en ambdós casos també és força diferent. Per exemple, un processador Atom pot consumir uns 10 watts, mentre que un processador Beckton pot consumir fins a 130 watts.

L'arquitectura

La figura 9 presenta les diferents etapes de la microarquitectura del Pentium 4, coneguda també com a *Netburst microarchitecture* (Intel). El *pipeline* consta de deu etapes diferents. Les quatre primeres són en ordre i són les relacionades amb la cerca i preparació de les instruccions (cerca, reanomenament i traducció a microoperacions). Les cinc següents poden ser fora d'ordre i són les encarregades de dur a terme l'execució funcional de les operacions. Finalment, la darrera s'executa en ordre i és on es finalitzen les instruccions.

Figura 9. *Pipeline* del Pentium 4



Durant les diferents etapes del processador hi ha recursos que es troben replicats per processador lògic, n'hi ha que es troben compartits però dividits per processador lògic i, finalment, n'hi ha que es troben totalment compartits.

En primer lloc, cada processador lògic manté una còpia separada del seu estat arquitectònic necessari per a la seva execució funcional correcta. Els recursos encarregats de guardar aquest tipus d'informació són els que es troben replicats per cada context:

- El punter a la instrucció següent.

- El *buffer* d'instruccions del fil, és a dir, el flux d'instruccions que el fil executarà potencialment.

- El *translation look-aside buffer* (TLB) usat per a fer la traducció d'adreces virtuals a físiques. Cada fil tindrà un mapatge d'adreça virtual a una adreça física diferent. Altrament, hi hauria problemes de seguretat potencials.

- El *return stack predictor* o predictor d'adreça de retorn. Com el seu nom indica prediu les adreces de retorn de les funcions.

- L'*advanced programmble interrupt controller* (APIC) orientat a la gestió d'interrupcions.

- La taula de reanomenament, necessària per a poder dur a terme el fora d'ordre i poder fer el remapatge de registres virtuals a registres físics.

Tot i que hi ha altres recursos que es troben replicats, els més importants són els esmentats anteriorment. També cal fer notar que aquesta descripció equival a una arquitectura *hyperthreading* general. Cada implementació concreta, per exemple la del processador Atom, pot tenir variacions en funció del tipus de requeriments que el processador té i del disseny que els arquitectes han dut a terme.

En segon lloc, trobem certs recursos que es troben dividits entre els diferents fils. En general, la majoria són *buffers*. Com, per exemple, el *reorder buffer*, els *load* i *store buffers*, cues entre les diferents etapes, etc.

Finalment, trobem altres recursos que es troben totalment compartits entre tots els fils del processador:

- La memòria cau d'instruccions o *trace cache* (i els *buffers* corresponents). Aquest és un mecanisme que es va dissenyar per a augmentar l'amplada de banda de l'etapa de cerca d'instruccions i reduir el consum del processador. Consisteix a guardar traces d'instruccions que ja s'han seleccionat prèviament i de les quals ja es té una descodificació.

- Les memòries cau.

- Els mecanismes d'execució fora d'ordre.

- Els predictors de salt.

- Lògica de control i busos de comunicació.

Un dels punts clau en un disseny SMT és la complexitat i àrea que s'afegeix al seu disseny pel fet de considerar els diferents fils d'execució. Un dels avantatges de l'*hyperthreading* és la capacitat de donar una millora de rendiment important

respecte d'un augment d'àrea reduït. Un dels motius principals d'això és que els processadors lògics comparteixen quasi bé tots els recursos del processador físic. L'augment d'àrea es deu bàsicament als estats arquitectònics extra, a la lògica de control addicional i a la replicació d'alguns dels recursos.

La perspectiva del sistema

L'*hyperthreading*, tal com s'ha introduït anteriorment, feia que un sol processador es veiés des del punt de vista del sistema com diferents processadors lògics (o fils d'execució). El processador té una còpia de l'estat arquitectònic per a cadascun dels processadors lògics, i tots els processadors lògics comparteixen un conjunt de recursos físics que permeten l'execució dels diferents fluxos d'instruccions.

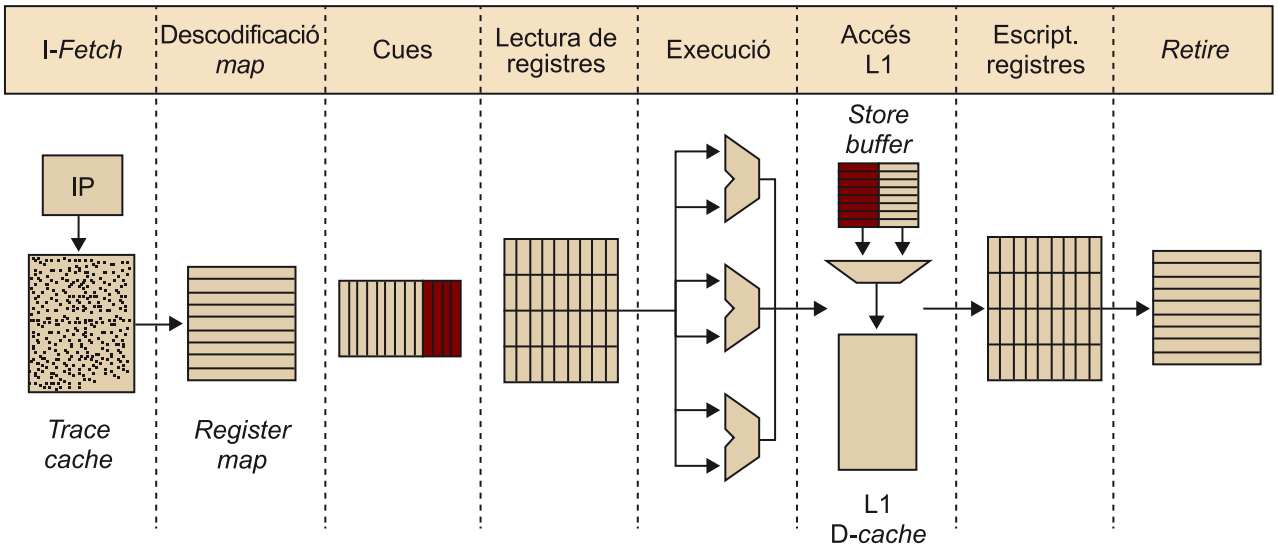
Des de la perspectiva del sistema operatiu i de les aplicacions, el maquinari els ofereix *N* processadors independents. El programari pot gestionar els diferents fils d'execució damunt dels processadors lògics en funció de les seves polítiques d'administració. El que és important fer notar és que, des del seu punt de vista, és equivalent a tenir a la seva disposició *N* processadors físics independents.

Des de la perspectiva del model de programació, l'*hyperthreading* es pot veure com una arquitectura multiprocessador amb temps d'accés uniforme a memòria (conegut com NUMA = *non uniform memory access*). Tots els fils tindran, en mitjana, un temps d'accés a memòria similar.

5.4.2. L'Alpha 21464

L'Alpha 21464 (Rusu, Tam, Muljono, Ayers i Chang, 2006), també conegut com a EV8, va ser l'evolució de l'Alpha 21364 (figura 10). Respecte d'aquest darrer, la millora més important de l'EV8 va ser la incorporació de tècniques SMT. Tot i que el seu rendiment estimat era força més elevat que el del seu predecessor, la línia de processador Alpha es va cancel·lar el 2001. Això va incloure la cancel·lació de l'EV8.

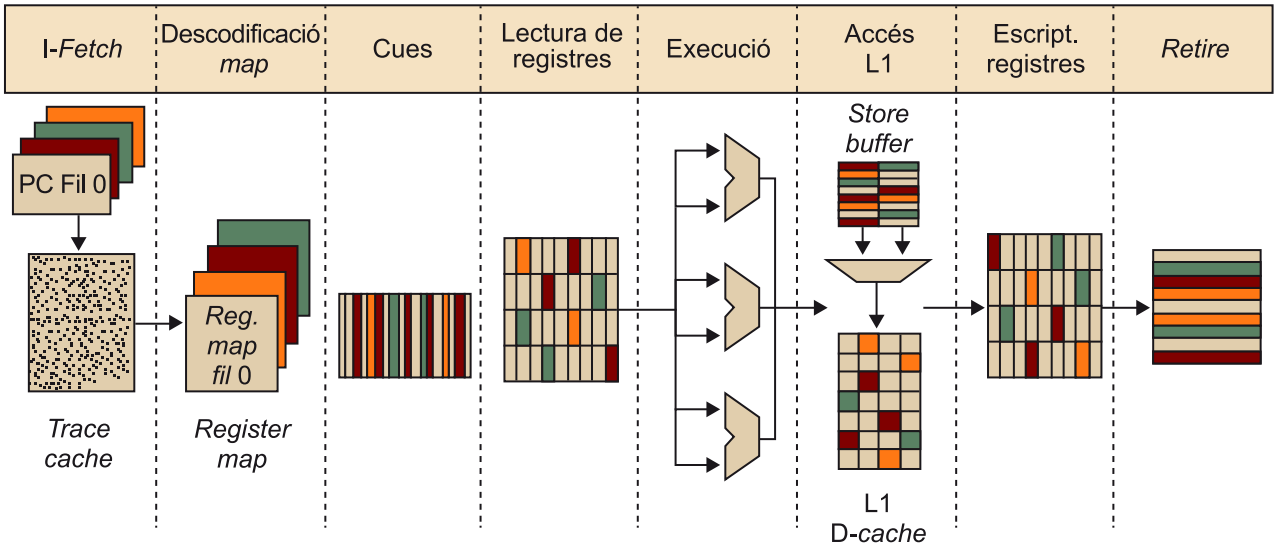
Figura 10. *Pipeline* de l'Alpha 21364



L'arquitectura

La figura 11 mostra les diferents etapes de què es componia. Per tal d'afegir SMT a l'arquitectura Alpha, EV8 inclou una nova lògica per a mantenir quatre comptadors de programes independents (un per fil). A cada cicle l'etapa de *fetch* selecciona un sol dels fils disponibles de la memòria cau d'instruccions. Un cop la instrucció és seleccionada i carregada, aquesta es marca amb l'identificador del fil al qual pertany. Aquest identificador anirà aparellat amb la instrucció durant totes les diferents etapes del processador (etapa de descodificació, unitats funcionals, etc.).

Figura 11. *Pipeline* de l'Alpha 21464



La lògica que gestiona la selecció fora d'ordre de les instruccions que poden ser executades pot escollir una instrucció de qualsevol dels quatre fils. De manera similar al que hem vist amb l'*hyperthreading*, la complexitat afegida a la lògica fora d'ordre i la lògica de reanomenament és força reduïda. En el cas de l'EV8 això implicava tan sols un increment del 6% de l'àrea.

El canvi més important introduït en les arquitectures Alpha per a donar suport a l'SMT va ser la incorporació de 32 registres enters i 32 registres coma flotant per a cadascun dels quatre fils d'execució. Per tant, desar tot l'estat arquitectònic de tot el processador necessita 256 registres. D'altra banda, per tal de poder dur a terme el reanomenament, i poder suportar un total de 256 instruccions en vol, l'EV8 disposa de 256 registres extra. En total, inclou un total de 512 registres.

A diferència de la microarquitectura NetBurst, l'Alpha 21364 només replica el comptador de programa i els *register map*. Tota la resta de recursos es troben compartits:

- La cua d'instruccions.
- El *register file*, que com s'ha esmentat es troba incrementat notòriament respecte de l'Alpha 21364.
- El primer i segon nivell de memòries cau.
- El *translation look-aside buffer* (TLB).
- El predictor de salts.

Des del punt de vista del sistema o l'aplicació, l'*hyperthreading* ofereix quatre fils d'execució totalment independents. Com s'ha comentat, cada fil es veu com un processador lògic diferent (quatre fils es veuen com quatre processadors independents).

La perspectiva del sistema

En el cas de l'EV8, el sistema també té accés a un sol processador físic. No obstant això, en aquest cas també hi ha un sol processador lògic. Cada fil és vist com una unitat de procés o *thread processing unit* (TPU), que comparteix recursos com ara la memòria cau d'instruccions, memòria cau de dades, el TLB o la memòria cau de segon nivell amb els altres fils.

5.5. Conclusions

Durant aquest apartat s'han estudiat diferents tècniques per a explotar el paral·lelisme a nivell de fil. En primer lloc, s'han presentat les arquitectures *super-threading*, en què les diferents etapes del processador són compartides pels diferents fils disponibles en el processador. Dins d'aquest tipus de compartició s'entra en detall en compartició de gra fi i de gra gruixut. Del darrer tipus s'estudia l'arquitectura de l'UltraSPARCT1.

En segon lloc, s'han estudiat les arquitectures de tipus *simultaneous multithreading*, en què els diferents fils del processador comparteixen les diferents etapes. Dins d'aquestes tècniques, s'han presentat diferents possibilitats de disseny i dues implementacions reals.

D'una banda, la implementació SMT d'Intel, l'Hyper-Threading. El processador inclou diferents processadors lògics (un per fil). Cadascun conté una còpia independent del seu estat arquitectònic (incloent-hi TLB, registres de reanomenament, comptador de programa, etc.) i comparteixen els recursos de les diferents etapes. En alguns casos, aquests recursos es troben dividits per fil (per exemple, cues o l'*store buffer*). En altres casos els recursos es troben totalment compartits pels fils (per exemple, registres de reanomenament).

De l'altra, s'ha presentat la implementació SMT d'Alpha, l'EV8. En aquest cas, el processador inclou diferents unitats de processament (també una per fil). Cadascuna conté una còpia separada dels seus registres i dels seus comptadors de programa. Ara bé, a diferència de l'Hyper-Threading, els fils de l'EV8 comparteixen tota la resta de recursos del processador (per exemple, accés a la L1, TLB, l'*store buffer*, etc.). Això implica que els fils no són independents i des del punt de vista del sistema veuen l'espai d'adreces.

Durant el proper apartat es presenten els límits del paral·lelisme que els SMT i els *hyperthreading* mostren quan es volen construir arquitectures amb un nombre més elevat de fils. Com a solució a aquests problemes d'escalabilitat, s'estudiaran les arquitectures multinucli. Aquestes arquitectures permeten escalar el nombre de fils de manera lineal sense topar amb increments tan costosos com en el cas de les arquitectures estudiades en aquest apartat.

6. Arquitectures multinucli

6.1. Limitacions de l'SMT i arquitectures *super-threading*

En tots els casos anteriors l'explotació del paral·lelisme es porta a terme afegint el concepte de *fil* a l'arquitectura del processador. En aquest cas, l'augment de paral·lelisme s'aconsegueix incrementant la lògica del processador, analitzant el diferent nombre de fils que es vol considerar.

Per exemple, si es volen tenir vuit fils, es replicaran vuit vegades les estructures necessàries (més o menys vegades en funció del disseny SMT que s'està considerant). Aquest model, tot i ser adequat i emprat en l'actualitat per molts processadors, té certes limitacions. A continuació, se'n discuteixen les més representatives.

6.1.1. Escalabilitat i complexitat

Els models SMT són adients per a un nombre relativament petit de fils (per exemple, dos, quatre o vuit fils). No obstant això, per a un nombre més gran de fils això pot comportar certs problemes d'escalabilitat. Com s'ha pogut veure en els apartats anteriors, per cadascun dels fils dels quals disposa un processador hi ha molta lògica que es troba replicada o compartida.

Aquest fet podria implicar tan sols un problema d'espai. És a dir, duplicar el nombre de fils implica duplicar el nombre d'entrades de l'*store buffer*, o duplicar el nombre de taules de reanomenament. Tot i així, l'increment en la quantitat i mida del nombre de recursos també té associat un increment exponencial en la lògica de gestió d'aquests recursos.

A tall d'exemple s'estudia el cas del *reorder buffer*. Aquesta estructura és l'encarregada de finalitzar totes les instruccions que ja han passat per totes les etapes del processador i que resten pendents de ser finalitzades (etapa de *commit*). En el cas de tenir dos fils d'execució, i assumint que es genera una instrucció per cicle per fil, cal poder finalitzar o realitzar el *commit* dues instruccions per cicle. Altrament, el sistema no és sostenible. Poder finalitzar dues instruccions per cicle és quelcom realista.

No obstant això, si s'incrementa el nombre de fils de manera lineal, no és realista esperar que es podrà construir un sistema real que sigui capaç de finalitzar el nombre proporcional d'instruccions necessàries per tal de mantenir-ne el rendiment.

La lògica i els recursos necessaris per a gestionar la finalització d'un nombre tan elevat d'instruccions en vol serien extremament costosos i probablement no assolibles (si es tinguessin 128 fils disponibles i 32 instruccions en vol per fil en caldrien 4.096). Si més no, considerant l'estat actual de la tecnologia de processadors.

6.1.2. Consum energètic i àrea

Per tal de donar suport a un nombre elevat de fils cal incrementar les estructures proporcionalment. En alguns casos aquests increments no són costosos en termes de complexitat i àrea, però algunes de les estructures són altament costoses d'escalar. Altra vegada, podem posar com a exemple l'accés a les memòries cau.

L'increment del nombre de ports de lectura o escriptura de les diferents memòries cau és altament costós (tant pel que fa a la complexitat com a l'àrea). Afegir un port de lectura nou en una memòria cau pot equivaldre a un increment d'un 50% de la seva àrea (Handy, 1998).

Com s'ha esmentat anteriorment, si es vol incrementar el rendiment de manera més o menys proporcional al nombre de fils cal també tenir-ho en compte en la manera com s'accedeix a la memòria cau. Per tant, si es volgués augmentar el nombre de fils, per exemple a 256 fils, caldria redimensionar (tant en àrea com en lògica) tota la jerarquia de memòria coherentment. Com es pot veure, i amb la tecnologia actual, això és impracticable.

El consum energètic d'un processador SMT que doni suport a un nombre molt elevat de fils és alt. En aquelles situacions en què no es fes ús de tots els fils disponibles o l'ús dels recursos corresponents fos ineficient, el consum del processador en watts seria molt elevat comparat amb el rendiment que se n'estaria obtenint.

Com s'analitza a continuació, en altres arquitectures i en aquestes situacions es pot aplicar *dynamic voltage scaling* (Yao, Demers i Shenker, 1995), és a dir, reduir la freqüència i el voltatge d'algunes parts del processador, ja que això permet reduir substancialment el consum del processador en situacions com la plantejada.

6.1.3. Producció

Durant les darreres dècades, atesa una gamma de processadors que segueixen un disseny arquitectònic similar (per exemple, els Sandy Bridge d'Intel), es treuen diferents versions d'un mateix processador. Atesa una mateixa família, es poden trobar versions orientades als clients (ordinadors d'ús domèstic), versions orientades a dispositius mòbils i versions orientades a servidors.

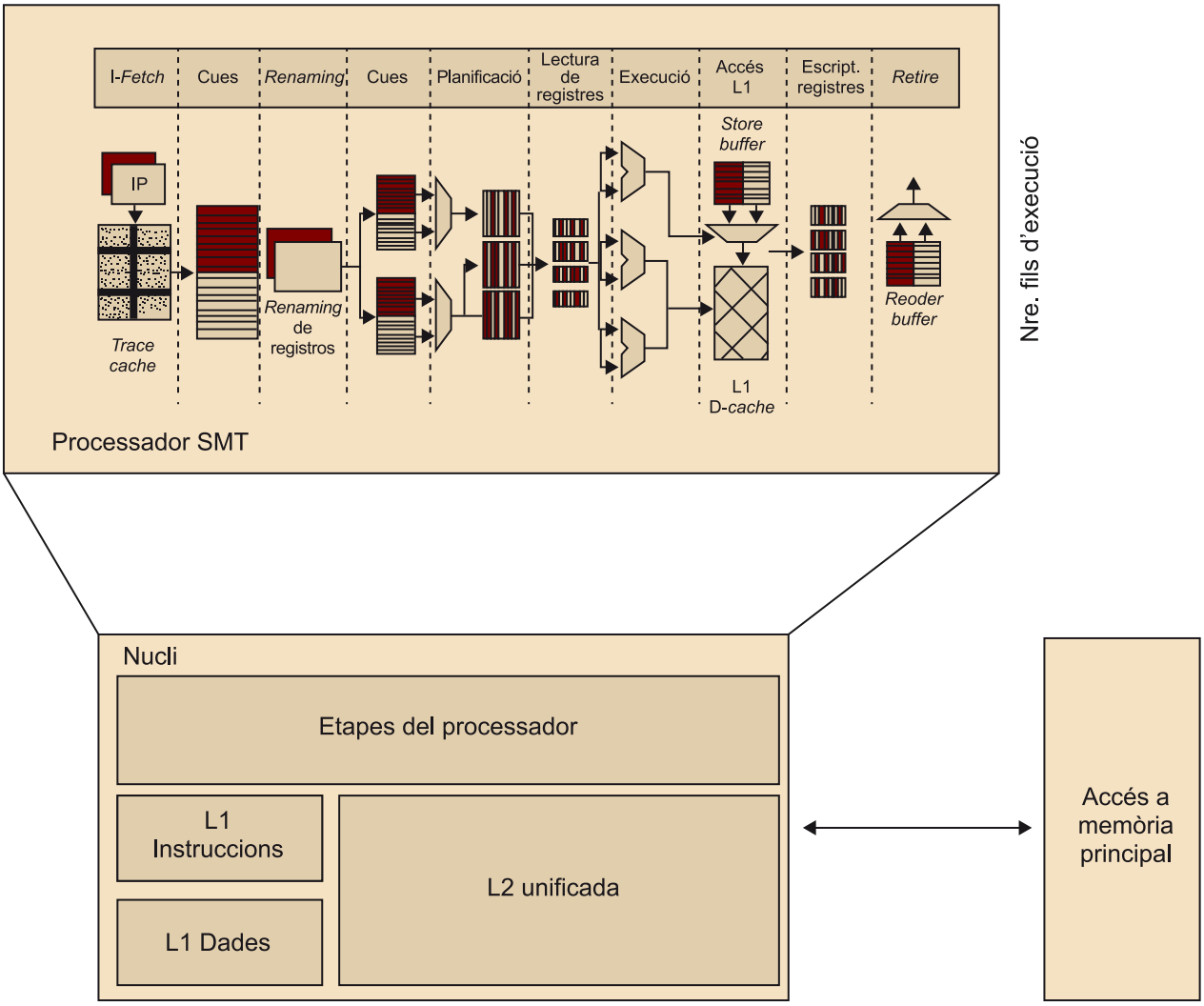
Per exemple, en el cas de la família Sandy Bridge es poden trobar els processadors domèstics amb dotze fils que consumeixen 130 watts, processadors per a dispositius mòbils de vuit fils que consumeixen 55 watts i processadors per a servidors amb setze fils i 150 watts de consum. Val a dir que no solament el nombre de fils varia entre els diferents tipus de processadors, sinó que també varien les mides de memòries cau i prestacions específiques (per exemple, la capacitat de connectivitat amb altres processadors). Dins d'un mateix tipus de processadors (per exemple, els clients), se'n troben moltes variants (per exemple, dins de la família Sandy Bridge de tipus client se'n poden trobar més de trenta variants).

Mirar de donar suport a tota aquesta varietat de nombre de fils limitant-se a escalar la quantitat de fils que l'arquitectura SMT suporta seria extremament costós des del punt de vista de producció. És a dir, la complexitat de tenir tants dissenys diferents encariria molt més el procés de disseny, producció i validació dels processadors. Com s'estudia en l'apartat següent, usant tecnologies multinucli aquest procés esdevé menys costós i més factible.

6.2. El concepte de *multinucli*

Durant els darrers apartats s'han estudiat diferents estructures multifil. Cadascuna implementava un processador superescalar fora d'ordre afegint-hi el concepte de *fil*. Independentment del tipus d'arquitectura multifil (*super-threading* o SMT), aquests processadors es podrien veure com un element de computació, amb n fils i una jerarquia de memòries cau. Aquesta abstracció (com es pot veure en la figura 12) es pot anomenar *nucli*. Cal remarcar que en aquest cas no inclou altres elements que un processador superescalar sí que inclouria: sistema de memòria, accés a l'entrada i sortida, etc.

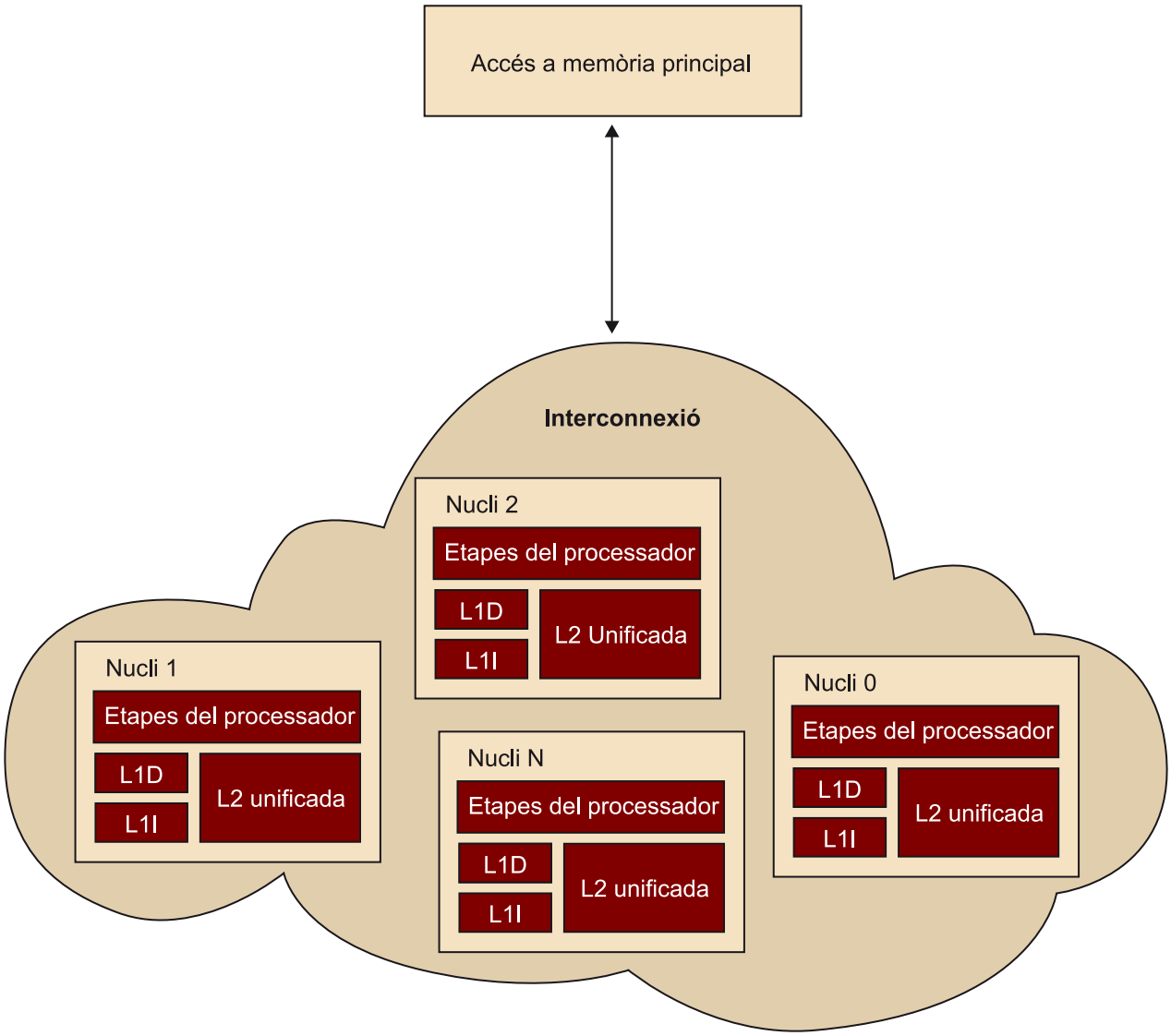
Figura 12. Abstracció de processador multifil



De fet, el concepte de *multinucli*, com el seu nom indica, consisteix a replicar *m* nuclis diferents dins d'un processador (figura 13). Cada nucli acostuma a tenir una memòria cau de primer nivell (de dades i instrucció), i pot tenir una memòria de segon nivell (acostuma a ser unificada: dades més instruccions). A part dels nuclis, el processador acostuma a tenir altres components especialitzats i ubicats fora d'aquests. Habitualment hi ha els següents: una memòria de tercer nivell, un controlador de memòria, components per a fer processament de gràfics, etc.

Tots aquests components (inclosos els nuclis) es troben connectats mitjançant una xarxa d'interconnexió. Aquesta és el mitjà físic i lògic que permet enviar peticions d'un component a un altre (per exemple, un petició de lectura d'un nucli a la L3).

Figura 13. Abstracció de multinucli



Com s'ha introduït, un dels components d'un multinucli fonamental és el controlador de memòria. Aquest gestiona les peticions d'accés al subsistema de memòria que fa la resta de components (tant lectures com escriptures).

Per si mateixa, una arquitectura multifil pot semblar senzilla. No obstant això, darrere d'aquest tipus d'arquitectures hi ha molta complexitat amagada que no es veu de manera directa: protocols de coherència, escalabilitat en la interconnexió, sincronització, desbalancejos entre fils, etc.

Durant els propers subapartats s'aprofundeix en els diferents conceptes que defineixen els multinucli. En primer lloc, es presenten algunes variants de l'arquitectura del model que s'ha descrit. En segon lloc, s'estudien les dues característiques més importants que defineixen un multifil: arquitectures d'interconnexió i mecanisme de coherència. Finalment, es prestarà una de les arquitectures multinucli comercial disponibles al mercat.

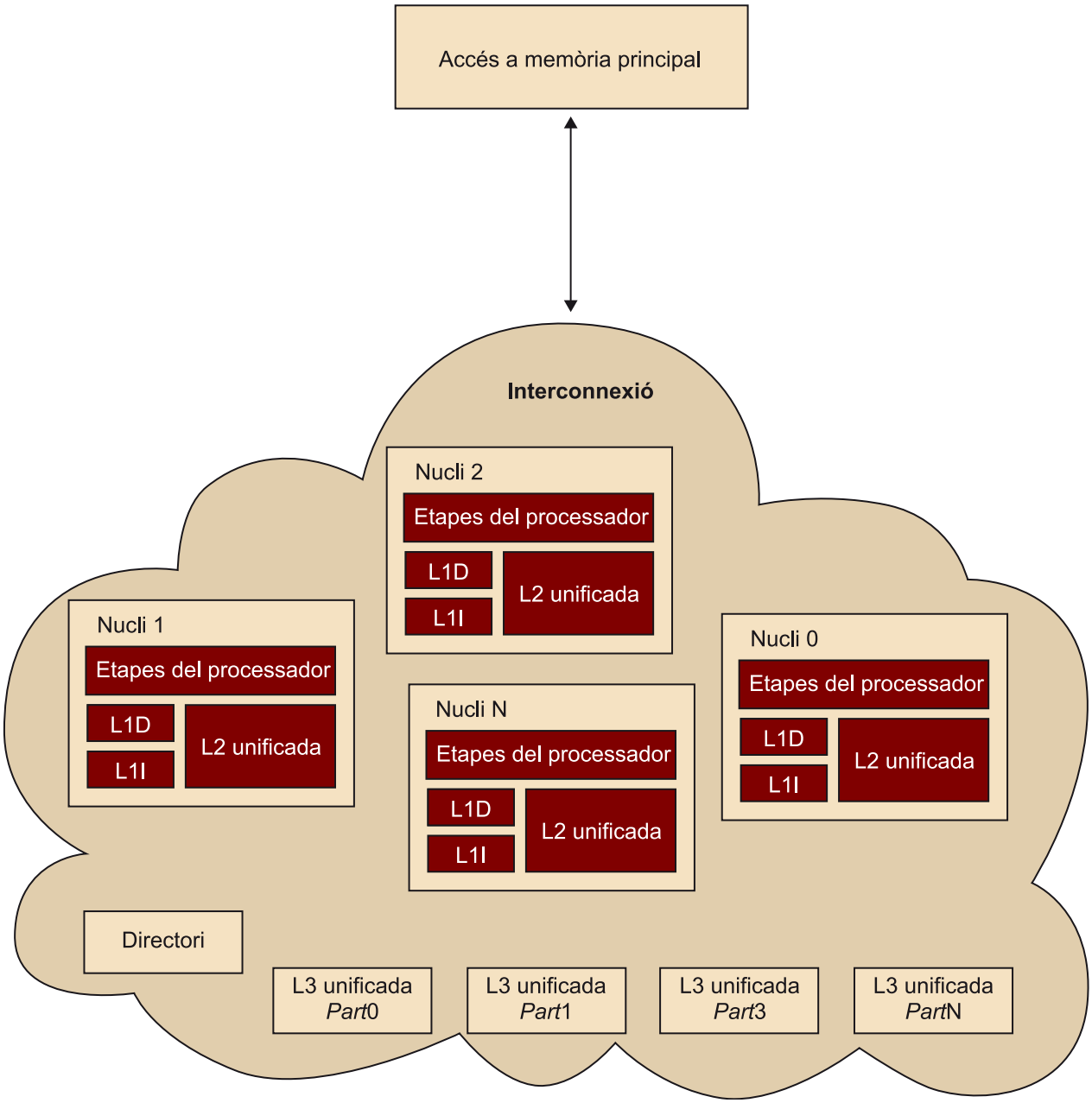
6.3. Variant de multinucli amb L3 i directori

El model discutit en el subapartat anterior descriu l'estructura més senzilla d'un multinucli: nuclis, interconnexió i accés a memòria principal. No obstant això, com es pot deduir, hi ha moltes variants d'aquest model.

La figura 14 mostra una arquitectura sovint emprada en el món de la recerca acadèmica i també present en models comercials. De la mateixa manera que el model anterior, s'hi troben un conjunt de nuclis que proporcionen accés a recursos computacionals (amb un fil o més) i a dues memòria cau (una de primera i una de segon nivell). A part d'aquests components, n'hi ha dos de nous: una memòria de darrer nivell (o memòria cau de tercer nivell) i un directori.

A diferència de la L1 o L2, aquesta L3 es troba dividida en blocs de la mateixa mida i ubicada en components separats. Com es presenta a continuació, el directori és l'encarregat de saber quines línies té cada bloc de la L3 i en quin estat es troben. D'altra banda, s'encarrega de coordinar i gestionar totes les peticions que els diferents nuclis fan a aquest darrer nivell de memòria cau.

Figura 14. Arquitectura multifil amb L3 i directori



En aquest model, quan una petició de lectura o escriptura no encerta cap línia ni de la L1 ni de la L2, la petició es reenvia a la memòria de tercer nivell. Conceptualment, això és equivalent al flux de fallada de la L1 que demana la mateixa petició a la L2. No obstant això, com s'ha comentat, aquesta petició es reenvia cap al directori.

El directori, atesa l'adreça que s'està demanant, té informació per tal de saber quin dels components de la L3 té aquesta línia i en quin estat es troba. En cas que cap component no la tingui, s'encarregarà de demanar-la a memòria, guardar-la en el bloc corresponent de la L3 i enviar-la al nucli. Com es veurà més endavant, això es fa a partir de protocols concrets que asseguren l'ordre i finalització en el tractament d'aquestes peticions.

En el cas anterior, s'ha assumit l'escenari en què ni la L1 ni la L2 del nucli contenen l'adreça que el fil del mateix nucli està demanant. També s'ha assumit que el nucli genera una petició de lectura al directori i aquest gestiona la petició a la L3. Tot i així, què passaria si algun altre nucli tingués la mateixa adreça a la seva L2/L1?

En aquest cas, podria passar el que s'està mostrant en la taula 7. El primer nucli llegeix l'adreça @X, la modifica i l'escriu de tornada a la memòria cau de darrer nivell amb el valor nou. Si durant tot el procés d'aquesta transacció un altre nucli llegeix el valor de @X de la L3, rebrà un valor incorrecte. En el cas de l'exemple el nucli 2 rebria un valor erroni.

Taula 7. Exemple de flux de lectures i escriptures incoherent

instant (n)	nucli 1	-->	lectura	@X =11	(L3 → L1/L2)
instant (n+1)	nucli 1	-->	escriptura	@X= 0	(L2)
instant (n+2)	nucli 2	-->	lectura	@X=11	(L3→ L1/L2)
instant (n+3)	nucli 1	-->	escriptura	@X=0	(L2→ L3)

Per a evitar aquests problemes es fan servir protocols de coherència. Aquests estan dissenyats per a evitar situacions com la presentada i mantenir la coherència entre tots els nuclis del processador.

6.4. Disseny d'arquitectures multinucli

En l'apartat anterior s'han introduït dues arquitectures multifil, i també els seus aspectes més rellevants. En aquest apartat es presenten dos aspectes representatius d'un sistema multifil i que defineixen en bona part el seu rendiment: la interconnexió i els mecanismes de coherència.

6.4.1. Interconnexions

Dins l'àmbit acadèmic s'han proposat moltes maneres diferents de connectar els diferents elements o components que componen un processador multinucli. La majoria d'aquestes propostes provenen de recerca ja feta en l'entorn de computació d'altes prestacions o *High Performance Computing* (HPC).

En aquest àmbit, el problema de connectar diferents components també apareix, però a una escala més gran. És a dir, en comptes de connectar nuclis, es connecten processadors entre ells o clústers. L'entorn HPC té molt més rodatge en la recerca d'aquest tipus de problemes atès que és una ciència que treballa en aquests problemes des de ben entrada la dècada de 1980, quan els primers ordinadors HPC van ser dissenyats.

De xarxes d'interconnexió, se n'han proposat de molts tipus (Agrawal, Feng i Wu, s/d), des de simples busos fins a estructures tridimensionals complexes com les topologies *Torus* o el *crossbar*. No obstant això, per temes de comple-

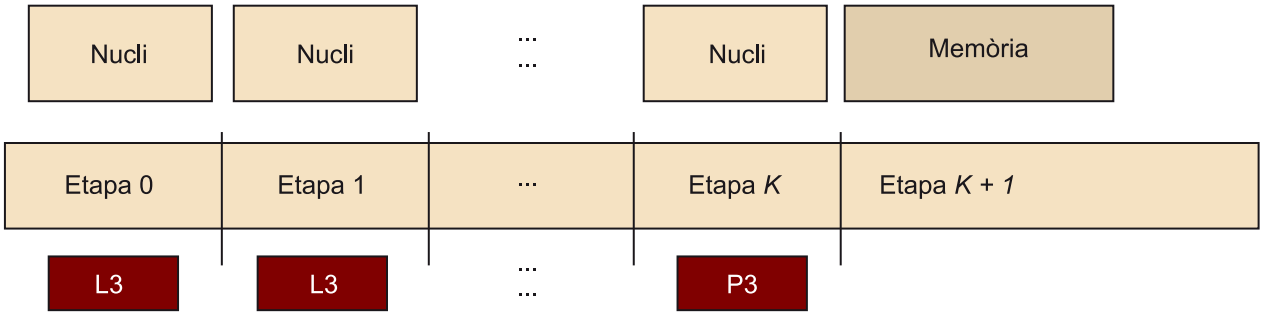
xitat o cost, no totes són aplicables al món dels multinuclis en què l'escala i restriccions de consum i àrea són més grans. Val a dir que com més avança la tecnologia, més complexes són les xarxes i més a prop d'aquests models complexos es troben els multinuclis.

A continuació, es presenten tres dels diferents models de xarxes d'interconnexió. És important remarcar que la bibliografia en aquest àmbit és molt extensa. En aquest apartat es farà una introducció als models més bàsics i es discutiran les característiques i els problemes més generals que tenen. Per a un aprofundiment més gran, és necessari analitzar les publicacions referenciades.

Xarxa tipus bus

Aquest tipus de xarxa es caracteritza perquè usa un sol bus bidireccional en què només es poden enviar transaccions d'un punt a un altre durant un interval de temps (Ceder i Wilson, s/d). És un bus compartit, per tant, si un component (com un nucli lateral) vol enviar informació a un altre (per exemple, la memòria), cap altre component no pot injectar dades al bus. Tots els nuclis es troben connectats a un àrbitre que gestiona l'accés al bus.

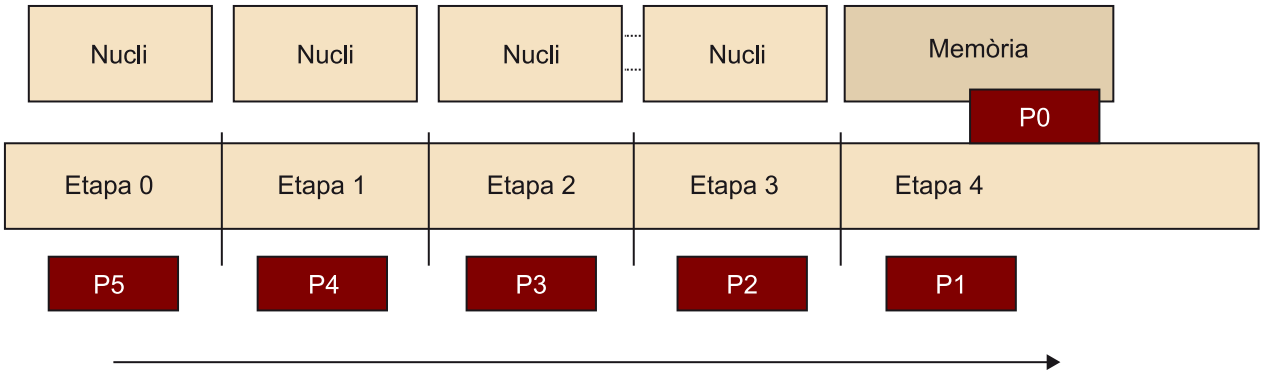
Figura 15. Interconnexió tipus bus



El bus té una capacitat de *k* bits. Per tant, si un component vol enviar una transacció que ocupa més de *k* bits dividirà les dades que cal enviar en *m* paquets (en anglès *flit*). D'altra banda, en la majoria de casos, no és possible enviar un paquet entre dos components en un cicle. Per tant, en aquestes ocasions, el bus es troba segmentat. Un paquet tardarà diversos cicles a arribar al destinatari. Tot i així, com que es troba segmentat, es podrà injectar un paquet per cicle. La figura 16 en mostra un exemple: el nucli zero envia una transacció que ocupa sis *flits*. Cada *flit* tarda quatre cicles a arribar a memòria, i la transacció triga un total de nou cicles a arribar sencera a memòria (el darrer paquet s'injecta sis cicles més tard que el primer).

Quan un component del processador vol enviar una transacció o dada a un altre component, abans cal que agafi la "propietat" del testimoni del bus. Un cop n'agafi el testimoni, pot emprar el bus durant *n* cicles consecutius (no fixats). Cal tenir present que una transacció pot necessitar un nombre arbitrari de paquets. Un cop ha acabat, tornarà el testimoni a l'àrbitre.

Figura 16. Transmissió d'una transacció del nucli a la memòria



L'avantatge d'aquest tipus d'interconnexió és la simplicitat. No obstant això, el rendiment respecte a les transaccions per segon que es poden enviar és molt baix. Un dels altres problemes que presenta és la falta d'escalabilitat. Aquesta arquitectura pot funcionar per a un nombre relativament baix d'elements. Ara bé, si es volen construir sistemes amb molts nuclis i components (blocs L3, directori, etc.) l'amplada de banda que el bus pot donar no escala prou. En aquests casos, les peticions dels nuclis experimentaran latències molt altes, ja que tots els altres components també estaran intentant accedir al bus. Hi ha certes variants d'aquesta arquitectura que donen més escalabilitat, com és el cas dels multibusos (Reed i Grunwald, 1987), que donen més rendiment que la presentada en aquest subapartat.

Com veurem, els dos tipus de xarxes següents, tot i ser més complexos, permeten treure molt més rendiment i són més escalables.

Xarxa tipus anell

Un bus tan sols permet accedir a dos components al canal de comunicació en un mateix instant de temps (Hong i Payne, 1989). Una extensió trivial d'aquest model és definir un bus que permeti injectar a tots els components de manera simultània.

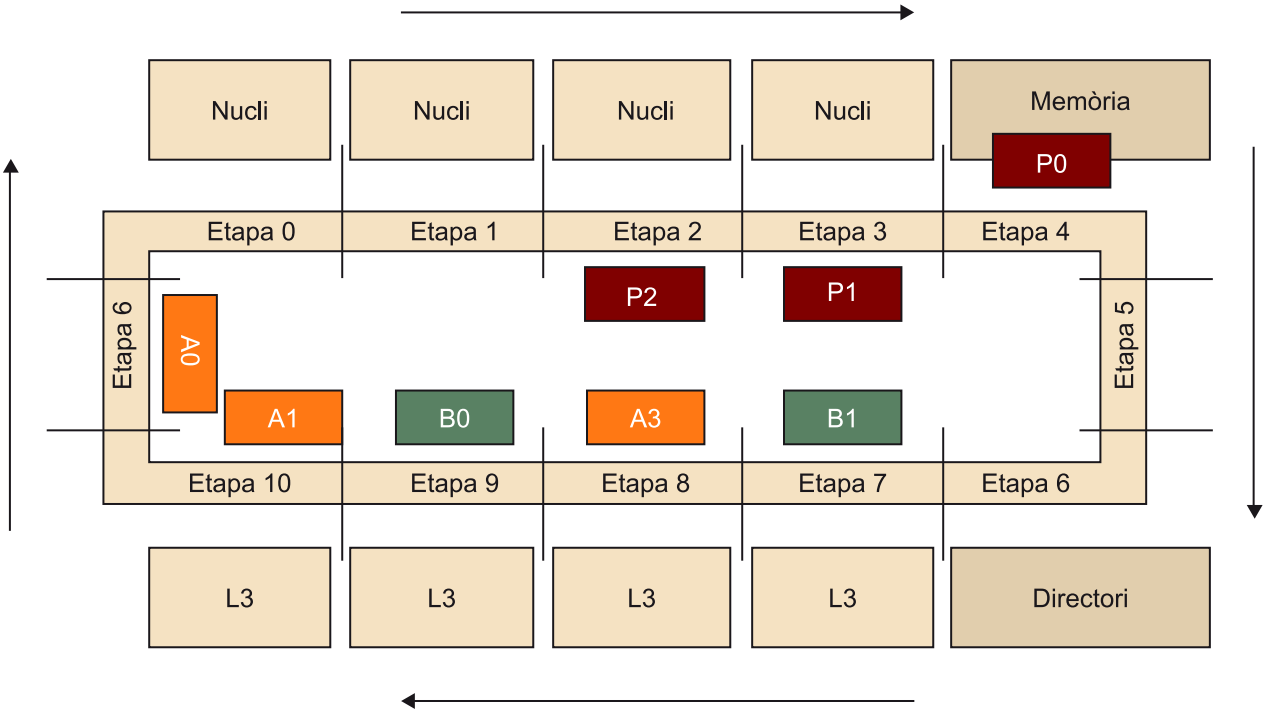
No obstant això, per construcció, això té problemes tant físics com lògics. Per exemple, si dos components volguessin enviar paquets de manera creuada això no seria possible, atès que els seus senyals toparien en el medi físic i el resultat seria quelcom indeterminat.

Una manera de fer més escalable l'arquitectura presentada anteriorment és connectant els extrems dels busos. En aquest cas, com es pot observar en la figura 17, la xarxa resultant té forma d'anell i els paquets circulen sempre en una

mateixa direcció per tal d'evitar aquestes col·lisions. Per una banda, la direcció tant pot ser en el sentit de les agulles del rellotge com el contrari. Per l'altra, l'anell es troba segmentat en diferents etapes, de manera que diferents paquets de diferents transaccions poden circular a la vegada. Els diferents components del processador es troben connectats a una de les etapes de l'anell i podran injectar a la xarxa sempre que l'etapa corresponent es trobi lliure.

En aquest tipus d'arquitectura, quan un component vol enviar una transacció (dividida en diferents paquets) a un altre component ha de verificar que l'etapa de l'anell en la qual es troba ubicat no es troba ocupada per cap altre paquet. Per exemple, en la figura 17, es poden observar tres paquets de la transacció P que es mouen del nucli 2 a memòria (P0, P1 i P2) i quatre paquets de dues transaccions que van de memòria als nuclis 1 i 2. Cal remarcar que en aquest cas el nucli 3 no podria injectar cap paquet, i que els paquets de la transacció A i B es troben intercalats.

Figura 17. Arquitectura tipus anell



Un dels avantatges clars d'aquest tipus d'arquitectures respecte d'un bus és l'increment de l'amplada de banda disponible de manera substancial. En aquest cas, totes les etapes de la xarxa poden ser usades pels diferents components connectats a l'anell. D'aquesta manera, s'aconsegueix reduir significativament la latència de les transaccions que els diferents nuclis volen injectar.

De la mateixa manera, cal comentar que aquesta interconnexió és més escalable que el bus. Tal com s'ha esmentat anteriorment, l'amplada de banda d'un bus ràpidament es queda limitada, en canvi, una estructura en anell té molta més escalabilitat.

Un exemple de processador comercial que usa aquesta arquitectura com a disseny d'interconnexió és l'Intel Sandy Bridge (Intel). Aquest té un total de quatre nuclis, quatre memòries cau L3, un controlador de memòria i un component de procés gràfic. Tots ells es troben connectats amb una interconnexió de tipus anell.

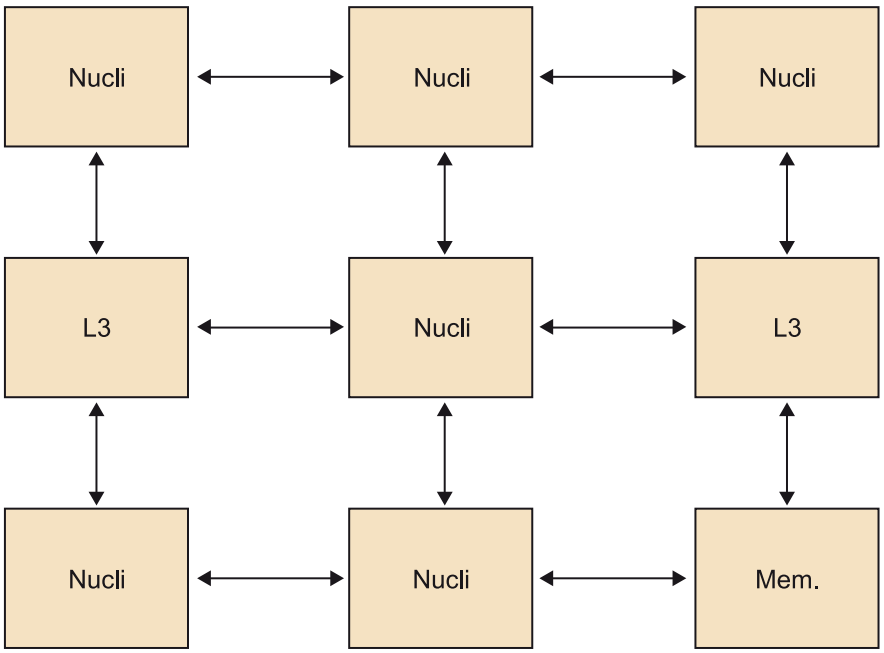
En aquest subapartat s'ha presentat un anell unidireccional. Ara bé, en l'àmbit acadèmic s'han proposat altres solucions que encara donen més rendiment, com, per exemple, l'anell bidireccional. Aquesta extensió permet enviar els paquets en ambdues direccions: en les agulles del rellotge i en sentit contrari. D'aquesta manera, s'està multiplicant l'amplada de banda de l'arquitectura per dos i es redueix el temps de viatge dels paquets de manera notòria. Tot i així, aquest descens no és gratuït perquè es necessita el doble de cables per a enviar la informació. Per tant, cal tenir en compte que això implica el doble d'àrea i el doble de consum energètic.

Xarxa tipus malla

L'increment d'escalabilitat, amplada de banda i reducció de latència d'un bus a un anell és substancial. No obstant això, en sistemes de dimensions molt grans fins i tot un anell pot no escalar. Alguns estudis acadèmics (Bell i altres, 2008) han analitzat la possibilitat d'emprar models altament escalables quan tenim requeriments molt elevats. Un exemple n'és la malla o *mesh*.

Com es pot observar en la figura 18, els diferents components del processador es troben connectats en forma de malla, és a dir, tenen connexions en l'eix de les Y i de les X. Cada component es connecta a la malla amb un encaminador o *router*. Aquest s'encarrega de dirigir els paquets que li vénen dels encaminadors als quals està connectat, als encaminadors adequats seguint una política d'assignació de ruta concreta.

Figura 18. Arquitectura tipus malla



Les malles tenen molta escalabilitat i donen una amplada de banda molt elevada. De fet, se'n poden trobar moltes variants atès que, per exemple, hi ha molts tipus de polítiques d'assignació de rutes. Ara bé, aquestes arquitectures són altament complexes i a causa d'aquesta complexitat s'hi troben problemes altament difícils de solucionar, com és el cas dels interbloquejos (*deadlocks*).

6.4.2. Coherència

Al principi d'aquest apartat s'ha introduït el concepte de *coherència* entre nuclis. Aquests mecanismes permeten que els accessos a memòria siguin coherents entre ells en tot moment. Per tant, sempre que un nucli està llegint o escrivint una dada es pot assegurar que és la seva darrera versió.

En aquest subapartat es presenten els tipus de mecanismes de coherència proposats, i també un exemple.

Protocols de coherència

Els protocols de coherència han de garantir que en tot moment la visió global de l'espai de memòria és coherent entre tots els nuclis. Atesa una línia de memòria @X:

- Si un dels nuclis en vol fer una lectura en rebrà la darrera còpia. No es pot donar el cas que un altre nucli l'estigui modificant mentre aquest en té una còpia.

- Quan un nucli demana una línia de memòria la pot demanar en exclusiva o en estat compartit. El nucli només podrà modificar la línia quan la tingui en estat exclusiu. Llavors, la línia estarà en estat modificat.

- Depenent del model de coherència que es faci servir, dos nuclis podran tenir la @X en les memòries cau en estat compartit. Ara bé, no podran modificar aquesta línia (ja que llavors tindríem dos valors diferents de @X en dos nuclis).

- Si un dels nuclis vol modificar la dada, abans s'haurà d'avisar la resta de nuclis que han d'invalidar aquesta línia. Si un altre nucli la torna a voler, primer s'haurà d'escriure a la L3 o a memòria i aquest l'haurà de llegir de L3. Depenent del protocol, en podem trobar certes variants.

Alguns dels punts anteriors dependran del tipus de model de coherència que estiguem considerant, especialment el tercer, ja que, per exemple, si el processador no suporta l'estat de línia compartit cada vegada que un nucli demana una línia, forçosament haurà d'invalidar aquesta línia de tots els nuclis que la tinguin.

En aquest apartat es considera un model MESI. En aquest cas, es considera que les línies de memòria que un nucli conté poden estar en un d'aquests quatre estats: *modified, exclusive, shared* i *invalid*. No obstant això, hi ha altres tipus de models (Stenström, 1990): MESIF, MOSI, MEOSI, etc.

El models de coherència es poden implementar sobre diferents tipus de protocols de coherència. En termes generals, es poden diferenciar en dues categories diferents:

1) Els protocols de tipus *Snoop*

En aquests protocols (Katz, Eggers, Wood, Perkins i Sheldon, 1985), quan un nucli vol fer alguna acció sobre una adreça de memòria, de lectura o d'escriptura, ha de dur a terme les notificacions pertinents a la resta de nuclis per tal d'obtenir-ne l'estat volgut. Per exemple, si vol tenir una adreça de memòria en exclusiva per tal de modificar-la, el nucli primer ha d'invalidar totes les còpies que els altres nuclis tinguin. Quan tots els altres nuclis hagin respost notificant que han processat la petició, el nucli ja podrà fer servir la línia. Abans, però, l'haurà de llegir de memòria o de la darrera memòria cau (L3).

2) Els protocols basats en directori

A diferència dels protocols de tipus *Snoop*, aquí els nuclis no s'encarreguen de gestionar la coherència quan volen usar una adreça de memòria (Chaiken, Fields, Kurihara i Agarwal, 1990). En aquest cas, apareix un component nou que s'encarrega de gestionar-la: el directori. Quan un nucli vol una línia en un

estat concret, la demana al directori. El directori acostuma a tenir una llista concreta dels nuclis que tenen l'adreça en qüestió i en quin estat la tenen. Per tant, quan processa una petició ja sap a quins nuclis ha d'avisar i a quins no.

El primer dels dos protocols és menys escalable que el segon i necessita molts més missatges per a gestionar la coherència entre nuclis. En el segon cas, el directori conté la informació de cada línia que els diferents nuclis tenen i en quin estat la tenen. Per tant, si un nucli llegeix una línia que cap altre nucli no té, el directori no enviarà cap missatge a la resta, sinó que directament li donarà la línia en estat exclusiu. En canvi, en el cas de protocols de tipus *Snoop*, el nucli enviarà missatges per a invalidar la línia concreta als diferents nuclis i en rebrà la notificació. En aquest segon cas el nombre de missatges que circularan per la xarxa serà molt més elevat que en cas del directori.

En qualsevol cas, el rendiment dels protocols basats en directori no és gratuït. Les estructures que guarden l'estat i els propietaris de les diferents línies és costosa tant respecte a l'àrea com al consum energètic. Per tant, per sistemes relativament petits serà més eficient emprar un sistema basat en *Snoop*.

Exemple: protocol de coherència

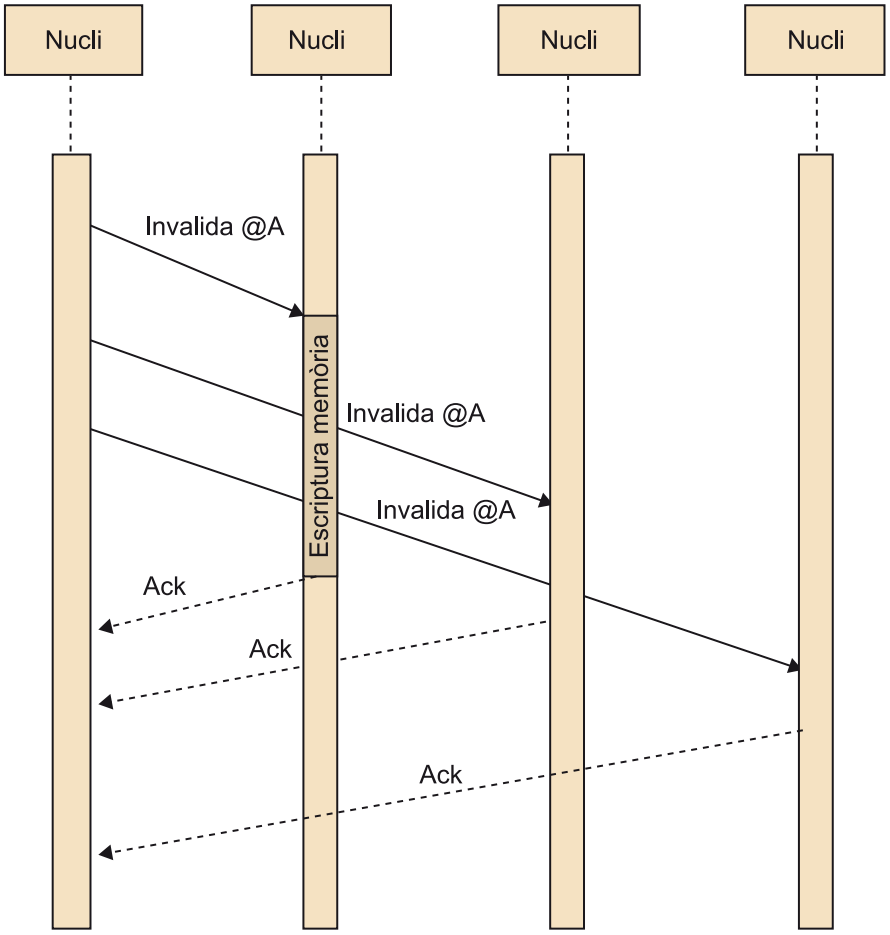
En aquest subapartat s'estudia un exemple de protocol de coherència basat en *Snoop*. Com a model de processador es considera: quatre nuclis, una memòria cau de tercer nivell dividida en blocs, un directori que controla quins components de la L3 tenen les adreces i un controlador de memòria. Per tal d'entendre com funciona aquest protocol, s'estudia el flux de missatges que genera cada operació que vol iniciar el processador. El protocol que s'explica és un protocol senzill. Aquest es podria millorar amb certes optimitzacions que es deixen com a exercici de l'estudiant.

Invalidació de còpies

Com s'ha esmentat en els darrers punts, quan un nucli vol llegir una línia de memòria per a modificar-la, abans ha de validar que cap dels altres nuclis no la tingui. Per tal de fer-ho haurà d'enviar *snoops* per a invalidar tots els altres nuclis que la tinguin (figura 19). Els nuclis que tinguin la línia en estat compartit o exclusiu, l'hauran de marcar com a invàlida en les seves L1 i L2.

En cas que un dels nuclis la tingui modificada l'haurà d'escriure a la memòria cau de darrer nivell i, llavors, invalidar-la en les seves L1 i L2. Cal recordar que si la línia es troba modificada, només la pot tenir un nucli. Depenent del tipus de política de memòria cau del processador, la línia s'escriurà a la L3 o a memòria: inclusiva / no inclusiva i *write-back/write-through* (Handy, 1998). En aquest exemple s'assumeix: inclusiva i *write-back*. Resta com a tasca del lector estudiar quines diferències hi ha entre aquests quatre tipus de polítiques.

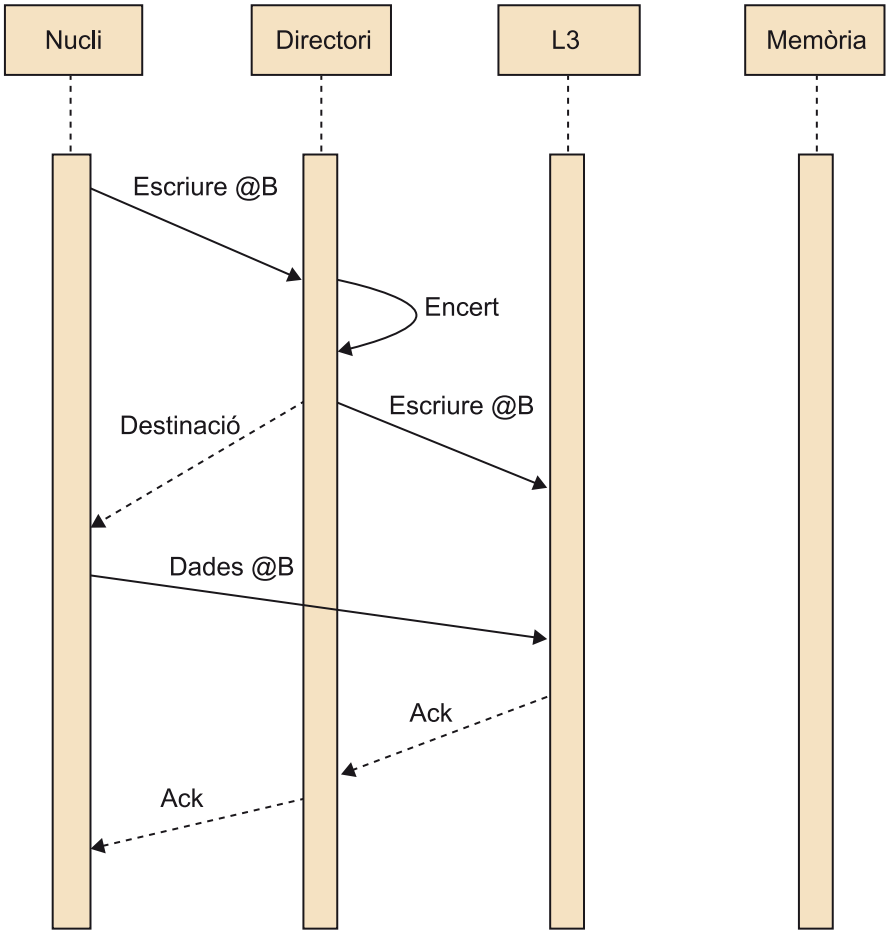
Figura 19. Invalidació de la línia @A



Escriptura a la memòria

Quan un nucli rep una petició d'una línia que té guardada en la seva L2 en estat modificat, l'ha d'escriure a la L3. Com es mostra a la figura 20, el nucli primer notifica al directori que vol escriure la línia a la L3.

Figura 20. Escriptura a memòria



El directori, usant les estructures internes, busca quin dels blocs de memòries cau L3 conté l'adreça @B. Un cop aquesta cerca ha finalitzat, notifica al nucli l'adreça del bloc en qüestió i notifica al bloc que rebrà dades corresponents a la victimització de l'adreça @B. Cal fer notar que aquest pas no és estrictament necessari, però sí que pot donar millor rendiment, ja que permet que la L3 prepari les estructures pertinents per a tramitar la transacció.

Un cop el nucli rep l'adreça on ha d'enviar la dada, genera els paquets necessaris cap a la L3 enviant les dades de @B. Quan la memòria cau rep les dades en fa l'escriptura al dispositiu. Un cop finalitzat, notifica al directori que ha acabat i el directori ho notifica al nucli. En aquest punt, les estructures relacionades a la transacció són alliberades del nucli i del directori.

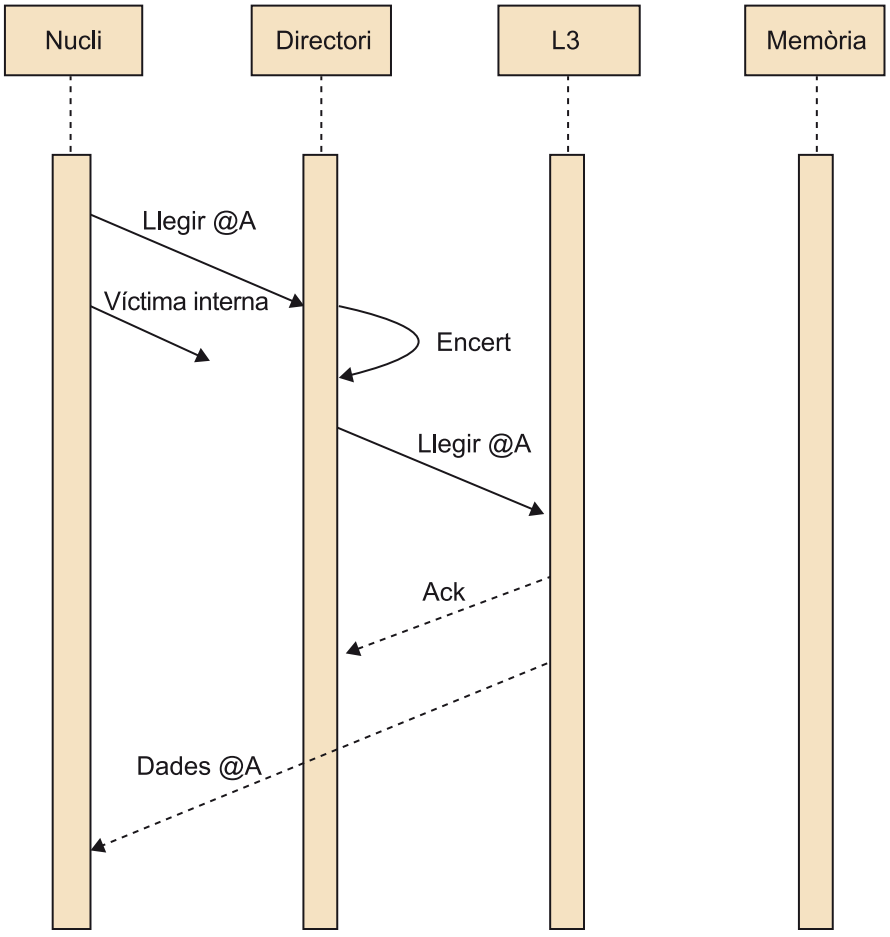
Lectura d'una adreça amb encert a la L3

La figura 21 presenta el flux de missatges que genera una transacció de lectura d'un nucli que encerta la memòria cau de tercer nivell. El nucli envia la petició al directori. Aquest, altre cop usant les estructures internes, cerca la línia amb adreça @A. Un cop finalitza la cerca, veu que ha encertat i del resultat extreu el bloc de memòria cau que té l'adreça en qüestió. El directori envia la petició de lectura al bloc. El bloc processa la transacció i envia les dades al

nucli i la confirmació que l'ha processat al directori. El directori allibera totes les estructures associades a la transacció un cop ha rebut la confirmació i el nucli escriu les dades a la seva L2.

Com es pot observar, al principi de tot el nucli ha generat una víctima interna. Sempre que es demana una dada a la L3, cal alliberar una entrada de la mateixa L2 (un *way* i un *set*; Handy, 1998). Altrament no tindria espai per a guardar la dada que s'està demanant. Aquesta dada victimitzada, en cas d'estar modificada, genera una transacció d'escriptura cap a la L3 seguint el flux d'escriptura de la memòria. En cas que la línia en qüestió no estigui modificada, el nucli no generarà cap transacció cap al directori, simplement alliberarà els recursos de la L2 en els quals està guardada.

Figura 21. Lectura a la L3 amb encert



Lectura d'una adreça amb error a la L3

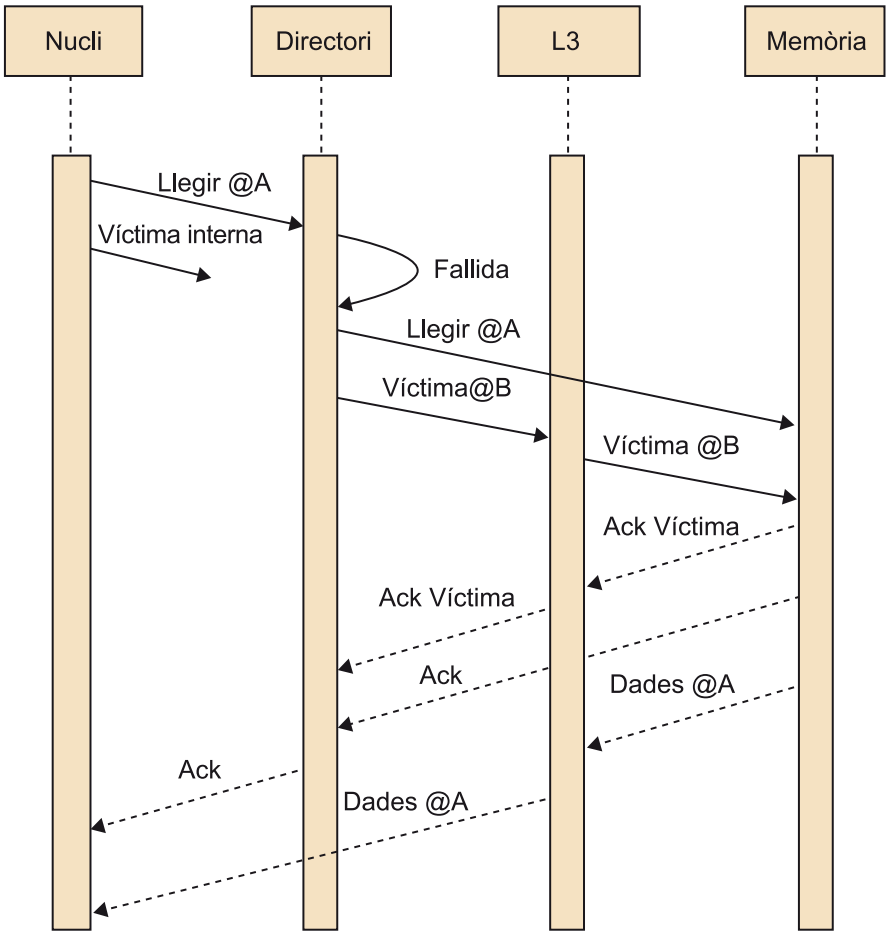
En el cas anterior s'ha vist el flux d'instruccions que genera una lectura en cas d'encert a la memòria cau de darrer nivell. En aquest cas, s'analitza què passa quan falla al directori.

Com es pot observar en la figura 22, el flux de missatges és exactament igual fins a l'arribada de la lectura al directori. Tant la lectura com la generació potencial d'una víctima interna a la L2 (per a fer espai per les dades de @A) són comunes.

Ara bé, en aquest cas el resultat de la cerca a la L3 és un error. Tanmateix, arribat aquest punt, el directori també ha pogut calcular quina de les L3 hauria de guardar les dades de @A. Cal recordar que assumint un model inclusiu de memòria, la línia @A haurà de ser present tant a la L2 del nucli que l'ha demanat com a la L3. Per tant, quan el directori demani les dades al component de memòria, aquest les haurà d'enviar tant a la L3 com al nucli.

Cal remarcar que de manera similar a la generació d'una víctima al nucli, per a escriure @A la L3 caldrà generar una víctima per tal de fer espai per a la nova adreça. Per tant, tal com es pot observar en la mateixa figura, el directori envia una transacció al bloc corresponent de L3 notificant que ha de victimitzar una línia concreta (@B) de la memòria cau per tal de fer espai per a @A. Un cop la L3 ha alliberat les estructures pertinents de @B, envia les dades a memòria.

Figura 22. Lectura a la L3 amb error



En paral·lel, el directori envia la petició de lectura al bloc de memòria per a l'adreça @A. Quan la memòria ha finalitzat el procés d'escriptura, enviarà la notificació de finalització a la L3 i aquesta al directori (això es pot fer de maneres diferents).

Un cop finalitzat el procés de lectura de @A, el bloc de memòria enviarà les dades al nucli i a la L3 de manera paral·lela. Així, la transacció de lectura acabarà quan les dades hagin arribat al nucli i el directori hagi notificat que també ha acabat la transacció.

6.5. Conclusions

Durant aquest apartat s'han analitzat les limitacions que les estructures SMT presenten. Sobretot s'ha destacat la complexitat i els problemes d'escalabilitat que podrien aparèixer si es volgués escalar el rendiment d'aquestes arquitectures a sistemes amb un nombre de fils més elevats (per exemple, 32 fils d'execució).

Com a evolució d'aquestes arquitectures, i continuant explotant el paral·lelisme a nivell de fil, s'ha presentat el concepte d'*arquitectura multinucli*. Aquest tipus d'arquitectura consisteix a replicar elements de procés diverses vegades dins d'un mateix processador. Aquests elements de procés poden ser a la seva vegada nuclis multifil que implementen les etapes d'un processador superescalar fora d'ordre SMT. Per tant, aquests sistemes poden tenir n elements de procés diferents i cadascun d'ells, m fils diferents. Per tant, es tindria un total de $n \times m$ fils. Aquest tipus d'arquitectures són molt escalables, ja que permeten augmentar el nivell de paral·lelisme sense augmentar-ne exponencialment la complexitat.

Ara bé, com s'ha pogut veure durant tot l'apartat, aquestes arquitectures també presenten reptes interessants. Els dos punts més crítics que s'han presentat per tal de poder tenir un sistema escalable i robust han estat les xarxes d'interconnexió i els mecanismes de coherència. El primer de tots és relatiu a la manera com els diferents components del processador es troben connectats. I el segon és relatiu a quins mecanismes té el processador per a assegurar que els accessos a dades de memòria siguin coherents entre els diferents nuclis.

La temàtica dels multinucli és altament extensa i complexa. Com s'ha apuntat durant tot l'apartat, s'han fet molts treballs acadèmics relacionats amb aquest àmbit. Aquest tipus d'arquitectures ha marcat tendències presents dins el mercat de processadors i dins l'àmbit de la recerca. Per tant, és interessant aprofundir en les diferents referències proporcionades per tal d'entendre més en detall els diferents punts tractats durant aquest apartat.

Resum

En aquest mòdul s'ha aprofundit en l'àmbit de les arquitectures multifil. En primer lloc, s'han analitzat les limitacions que presenten les arquitectures superescalars fora d'ordre amb un sol fil. Com s'ha vist, el rendiment que en podem extreure explotant el paral·lelisme a nivell d'instrucció es troba altament limitat pel tipus d'aplicacions que es volen executar. Per a algunes d'elles, tot i augmentar-ne infinitament els recursos, el rendiment es troba limitat per la seva característica inherentment no paral·lela. Tot i això, per a les aplicacions que més escalaven suposant recursos infinits hem pogut veure que el seu rendiment no augmentava de manera infinita.

Aquest estudi previ motiva l'aparició de les arquitectura multifil. En aquest tipus d'arquitectures el rendiment del processador augmenta incrementant el nombre de fils d'execució independents que aquest pot córrer. Aquestes exploten el que s'anomena *paral·lelisme a nivell de fil* (TLP). En aquest cas, augmentant el nombre de fils infinits podem aconseguir un processador teòric amb rendiment infinit, atès que aquests fils poden ser independents entre ells. No obstant això, tal com s'ha vist durant la resta d'apartats, aquest tipus d'arquitectures presenten altres complexitats i reptes.

Un cop explicada la motivació de les arquitectures, s'han presentat els diferents tipus multifil que s'han considerat més rellevants durant les darreres dècades. Començant per les arquitectures *super-threading*, seguint per les arquitectures *simultaneous multithreading* (SMT) i finalitzant per les estructures multinucli.

Com s'ha vist, les arquitectures SMT estenen el concepte de *processador fora d'ordre* afegint-hi el model fil. Els fils poden ser totalment independents, com en les arquitectures Intel d'*hyperthreading*, o bé compartir el mateix espai d'adreces com el cas de l'Alpha 21464. D'altra banda, poden compartir més o menys estructures de les diferents etapes (per exemple, el *reorder buffer*). En cas que les arquitectures no siguin compartides cada fil en té una còpia, de manera que n'obté un rendiment més elevat però també un consum i un espai molt més alt.

Les arquitectures SMT afegeixen una millora substancial respecte dels processadors que tan sols exploten el paral·lelisme a nivell d'instrucció. No obstant això, també s'ha pogut veure que la complexitat d'aquests dissenys és força elevada. A més a més, aquesta augmenta exponencialment en funció del nombre de fils que es volen facilitar. Per tant, l'escalabilitat d'aquest disseny és relativament limitada.

Com a evolució de les arquitectures multifil s'han discutit les arquitectures multinucli. Aquestes estenen el concepte de *processador* multiplicant el nombre d'unitats de processament. Aquest es troba compost per *n* diferents nuclis de procés, en què cadascun pot ser alhora un SMT o un *super-threading*. Els diferents components de processador es troben connectats amb una xarxa d'interconnexió i en general es comuniquen usant protocols de coherència. Aquests darrers asseguren que en tot moment tots els nuclis diferents tenen una visió coherent de l'espai d'adreces, és a dir, que atesa una adreça de memòria tots en veuen el mateix contingut i estat.

Durant la darrera part s'ha explicat quines són les característiques i quins són els punts de disseny més importants en aquestes arquitectures. S'han explicat algunes de les xarxes d'interconnexió i s'han discutit les característiques principals dels mecanismes de coherència. Val a dir que tant en l'àmbit acadèmic com en el comercial s'ha fet molta recerca en l'àrea dels multinuclis. Per tant, en aquest mòdul tan sols se n'han cobert alguns dels aspectes més rellevants. Per a aconseguir un coneixement més profund de cadascuna de les seves parts diferents, cal aprofundir en les referències proposades.

Activitats

1. Per tal d'aprofundir en les tècniques associades a la generació i selecció d'instruccions en un SMT llegiu l'article indicat a continuació. Cal enumerar les principals diferències i propostes respecte a les mateixes etapes d'un processador escalar fora d'odre SMT.

M. Tullsen. "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous-Multithreading Processor Dean".

2. Enumereu quinze característiques de l'Hyper-Threading que no s'hagin enumerat durant aquesta unitat didàctica. Com a font de dades feu servir la font següent:

http://software.intel.com/en-us/articles/introduction-to-hyper-threading-technology/

3. Una de les mètriques més importants que cal optimitzar en el disseny de processadors és el consum energètic. En cap dels apartats anteriors no hem entrat en detall en l'impacte de la complexitat dels processadors multifil o multinucli en el consum energètic. Enumereu les aportacions més interessants de l'article següent sobre això.

Benjamin Lee; David Brooks. "Effects of Pipeline Complexity on SMT/CMP Power-Performance Efficiency".

4. Proposeu una variant del protocol presentat en l'apartat "La perspectiva del sistema" suposant que en l'arquitectura considerada suprimim la memòria cau de darrer nivell i el directori. En aquest cas, només hi haurà memòria i nuclis.

Bibliografia

Agrawal, D. P.; Feng, T. Y.; Wu, C. L. (s/d). "A survey of communication processors systems". *COMPSAC* (pàg. 668-673).

Alverson, R.; Callahan, D.; Cummings, D.; Koblenz, B. (1990). "The Tera computer system". A: *International Conference on Supercomputing* (pàg. 1-6).

AMD (s/d). *Pàgina AMD Fusion*. Recuperat el 21 de desembre del 2012, de fusion.amd.com.

Bailey, D.; Barton, J.; Lasinski, T.; H., A. S. (1991). "The NAS parallel benchmarks". *Technical Report RNR-91-002 Revision 2, 2, NASA Ames Research Laboratory*.

Bell, S.; Edwards, B.; Amann, J.; Conlin, R.; Joyce, K.; Leung, V. i altres (2008). "TILE64 Processor: A 64-Core SoC with Mesh Interconnect". A: *IEEE International Solid-State Circuits Conference*.

Ceder, A.; Wilson, N. (s/d). "Bus network design". *Transportation Design* (pàg. 331-344).

Chaiken, D.; Fields, C.; Kurihara, K.; Agarwal, A. (1990). "Directory-based cache coherence in large-scale multiprocessors". *Computer* (pàg. 49-58).

Chase, J.; Doyle, R. (2001). "Balance of Power: Energy Management for Server Clusters". *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*.

Dixit, K. M. (1991). "The SPEC benchmark". *Parallel Computing* (pàg. 1195-1209).

Fisher, J. (1893). "Very Long Instruction Word Architectures and the ELI-512". *Proceedings of the 10th Annual International Symposium on Computer Architecture* (pàg. 140-150).

Handy, J. (1998). *The cache memory book*. Londres: Academic Press Limited.

Hong, Y.; Payne, T. H. (1989). "Parallel Sorting in a Ring Network of Processors". *IEEE Transactions on Computers* (vol. 38, fascicle 3).

IBM (2011). *AS/400 Systems*. Recuperat el 20 de desembre del 2011: http://www-03.ibm.com/systems/i/.

Intel (s/d). *Sandy Brid'intel*. Recuperat el 10 de desembre del 2011: http://software.intel.com/en-us/articles/sandy-bridge/.

Katz, R. H.; Eggers, S. J.; Wood, D. A.; Perkins, C. L.; Sheldon, R. G. (1985). "Implementing a cache consistency protocol". A: *Proceedings of the 12th Annual International Symposium on Computer Architecture*.

Kongetira, P.; Aingaran, K.; Olukotun., K. (2005). "Niagara: A 32- Way Multithreaded SPARC Processor". *IEEE MICRO Magazine*.

Lo, J. (1997). "ACM Transactions on Computer Systems". *Converting Thread-level Parallelism to Instructionlevel Parallelism via Simultaneous Multithreading*.

Marr, D. (2002). "Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History". *Intel Technology J.* (vol. 8, fascicle 1).

Melvin, S. (2003). "Flowstorm porthos massive multithreading (mmt) packet processor" [en línia]. <www.zytek.com/~melvin/flowstorm.html>.

Melvin, S. (2000). *Clearwater networks cnp810sp simultaneous multithreading (smt) core*. Recuperat el 19 de desembre del 2011, de www.zytek.com/~melvin/clearwater.html.

Rusu, S.; Tam, S.; Muljono, H.; Ayers, D.; Chang, J. (2006). "A dual-core multi-threaded Xeon(r) processor with 16 Mb L3 cache". A: *Proc. 2006 IEEE Int. Solid-State Circuits Conf.* (pàg. 315-324).

Seznec, A.; Felix, S.; Krishnan, V.; Sazeide, Y. (2002). "Design tradeoffs for the Alpha EV8 conditional branch predictor". A: *Proceedings of the 29th International Symposium on Computer Architecture*.

Song, P. (2002). "A tiny multithreaded 586 core for smart mobile devices". A: *2002 Microprocessor Forum (MPF)*.

Stenström, P. (1990). "A Survey of Cache Coherence for Multiprocessors". *IEE Computer Transactions*.

Tullsen, D. M.; Eggers, S.; Levy, H. M. (1995). "Simultaneous multithreading: Maximizing on-chip parallelism". *22th Annual International Symposium on Computer Architecture*.

Yao, E.; Demers, A.; Shenker, S. (1995). "A scheduling model for reduced CPU energy". *IEEE 36th Annual Symposium on Foundations of Computer Science* (pàg. 374-382).