UNIVERSITA' DEGLI STUDI DI PISA

FACOLTA' DI INGEGNERIA

Corso di laurea specialistica in Ingegneria Informatica

Tesi di laurea specialistica

Analisi e progettazione di tecniche di gestione per cache L2 Way Adaptable

Relatori:

Prof. Cosimo Antonio Prete
Prof. Piefrancesco Foglia
Ing. Alessandro Bardine

Candidato:

Diego Giardinetto

A.A. 2007/2008

Introduzione

La memoria cache è una particolare memoria che sfrutta la località spaziale e temporale dei dati per fornire un miglioramento del tempo medio di accesso ai dati da parte delle unità di elaborazione. Le memorie cache rivestono un ruolo importante all'interno degli attuali microprocessori. Infatti per ottenere un miglioramento delle prestazioni, negli ultimi anni si è assistito a un continuo aumento delle dimensioni della cache. Col passaggio dalle tecnologie microelettroniche a quelle nano elettroniche, però, si è avuto un aumento di potenza statica dovuto alle correnti di perdita (leakage) dei transistor.

Poiché tale consumo di potenza è proporzionale al numero dei dispositivi fisici (transistor) le cache di grandi dimensioni sono una delle maggiori fonti di consumo di potenza dei microprocessori. Quindi diventa di fondamentale importanza l'impiego di tecniche per il risparmio energetico per le memorie cache, focalizzate sul consumo di potenza statica.

Inoltre negli ultimi anni il progresso delle tecnologie microelettroniche ha reso possibile l'aumento delle prestazioni dei microprocessori: la continua diminuzione della feature size impiegati nei processi costruttivi dei circuiti integrati, ha permesso l'incremento del numero di transistor presenti all'interno dei singoli chip.

L'aumento del numero di transistor e delle frequenze operative, però, non è stato accompagnato da un progresso tecnologico significativo per quanto riguarda i collegamenti fisici all'interno dei circuiti integrati; per questo il ritardo introdotto dai fili (wire-delay), rispetto al periodo di clock, è aumentato [1].

Relativamente ai sottosistemi di memoria, per mitigare il problema del wire-delay, sono state introdotte la cache NUCA (Non-Uniform Cache Architecture). Le NUCA sono cache di secondo livello, che si differenziano dalle cache tradizionali (UCA, Uniform Cache Architecture) perché prevedono una matrice di banchi accessibili in maniera indipendente. Ogni banco è caratterizzato da un tempo di accesso che è funzione della posizione del banco stesso rispetto al controller e i banchi con minore latenza sono quelli che si trovano in prossimità di esso; inoltre ogni banco è di dimensioni più ridotte e permette quindi un accesso più rapido.

Le cache di tipo NUCA sono divise in categorie, identificate dalla capacità o meno dei dati contenuti nei banchi della cache, di migrare nei banchi adiacenti. Le memorie con questa abilità sono dette D-NUCA (Dynamic Non-Uniform Cache Architecture), mentre quelle che hanno un assegnamento statico dei dati nei blocchi sono dette S-NUCA (Static Non-Uniform Cache Architecture).
Le memorie D-NUCA way-adaptable sono una tipologia di cache L2 in grado di poter accendere o spegnere parti della memoria non strettamente necessaria, al fine di ridurne il consumo di potenza [6, 8].
L'obiettivo del presente lavoro di tesi è stabilire l'ottimalità dei parametri di funzionamento dell'algoritmo way-adapting differenziale, utilizzato per la gestione delle cache D-NUCA Way-Adaptable e descritto nella tesi di laurea [7]. Durante lo studio dell'ottimalità di tali parametri, l'analisi delle grandezze associate alle riconfigurazioni della cache ha portato all'identificazione di alcuni comportamenti ricorrenti, che ha permesso la progettazione di una nuova famiglia di algoritmi che non prevede l'utilizzo delle soglie, permettendo di svincolarsi dalla necessità di dover calcolare e specificare una coppia di soglie.
Gli algoritmi così identificati sono confrontati, in termini di IPC, associatività media e consumo di potenza, su un insieme di benchmark delle suite SPEC e NAS.
Il documento è organizzato come segue:

- *Capitolo 1*: viene descritto lo stato dell'arte riguardo le memorie cache UCA e NUCA;
- *Capitolo 2*: viene descritta la metodologie di esplorazione dell'albero delle riconfigurazioni (tuning) e le rispettive metriche di esplorazione;
- *Capitolo 3*: viene presentata la tecnica di estrazione delle soglie e due tecniche alternative che permettono di velocizzarne l'estrazione (ottenendo una prima approssimazione) in caso di un numero elevato di vincoli;
- *Capitolo 4*: si evidenziano gli strumenti utilizzati nel lavoro di tesi;
- *Capitolo 5*: vengono presentati i risultati ottenuti.

1. Stato dell'arte

Nel presente capitolo verrà trattata una panoramica generale delle tecnologie attualmente esistenti per la realizzazione di memorie cache di secondo livello (L2). Procedendo verso tecnologie più recenti e complesse, saranno esaminate prima le memorie di tipo UCA (Uniform Cache Architecture) e successivamente quelle di tipo NUCA (Non-Uniform Cache Architecture). Queste ultime sono a loro volta suddivise in cache statiche (S-NUCA) e cache dinamiche (D-NUCA); tale architettura dinamica sarà oggetto di approfondimento [1]. Si introdurrà, inoltre, il concetto di cache D-NUCA way-adaptable, che risulterà indispensabile nell'esposizione di questo lavoro di tesi.

1.1 Memorie cache UCA e ML-UCA

Le memorie cache di tipo tradizionale (UCA, Uniform Cache Architecture) sono formate da un unico banco di memoria SRAM, suddiviso in blocchi la cui dimensione dipende dalla quantità di informazione trasferibile tra due livelli di memoria adiacenti. Questo tipo di memorie, sono affette da problemi relativi alla latenza media di accesso ai dati, visto che il tempo di accesso ad ogni blocco è determinato dal ritardo delle interconnessioni, che obbliga a dover attendere che vengano acceduti i banchi più remoti, anche nel caso in cui i dati fossero già pronti.

Le cache ML-UCA (Multi-Level UCA) affrontano il problema della latenza di accesso organizzando la memoria in una matrice di banchi, il cui tempo di accesso individuale è calcolato in base a quello del blocco più lontano dalla porta di ingresso della cache, penalizzando però di fatto i banchi più fisicamente vicini, attribuendo loro una latenza sovradimensionata.

Sebbene l'architettura multilivello permetta più accessi contemporanei evitando conflitti tra i vari accessi, sono meno performanti e più costose a causa dell'inevitabile incremento della superficie del chip. Inoltre, dovendosi attenere al principio di inclusione, esiste un considerevole problema di ridondanza dei dati, che non permette un uso efficiente dello spazio utilizzato.

Il principio di inclusione richiede infatti che ogni livello superiore della cache debba contenere tutti i dati presenti nel livello inferiore [1, 12].

Le cache in questione sono associative a insiemi.

Tech. (nm)	L2 size (MB)	N° sub-banks	Unloaded latency	Loaded latency	IPC	Miss rate
130	2	16	13	67.7	0.41	0.23
100	4	16	18	91.1	0.39	0.20
70	8	32	26	144.2	0.34	0.17
50	16	32	41	255.1	0.26	0.13

Figura 1.1. Performance di una cache UCA

L'unloaded latency è il tempo medio di accesso (in cicli di clock) alla cache, assumendo di non essere in presenza di conflitti, a differenza del loaded latency che invece ne prevede la presenza. Quest'ultima viene calcolata in modo sperimentale con l'ausilio di strumenti di simulazione. La terza colonna esprime invece la suddivisione in sottobanchi di ogni banco della cache.
E' possibile notare come, al crescere della dimensione della cache, la latenza sia già alta in assenza di conflitti e come questi possano incidere seriamente sulle prestazioni (in media un accesso ad una cache di 16MB impiega 255 cicli di clock). Un conflitto può essere causato da due accessi diretti allo stesso banco (bank contention) oppure da due accessi che devono usare la stessa connessione (canned contention).
Per ogni cache di grandezza diversa è inoltre riportato l'IPC trovato con il simulatore e la percentuale di miss in L2. E' facile notare come l'IPC decresca sensibilmente al crescere della dimensione della cache a causa proprio del significativo aumento della latenza.
Risulta dunque chiaro come questa architettura non possa essere applicata efficacemente per cache di elevata capacità, o comunque per memorie in cui la latenza assuma un aspetto non trascurabile (wire-dominated cache).

1.2 Memorie cache NUCA

Le cache di tipo NUCA, realizzano l'architettura della memoria come una matriche di banchi, in cui è possibile accedere a quelli più vicini senza dover aspettare che vengano acceduti quelli più remoti.. Per realizzare questo meccanismo, è però necessario che venga introdotta una rete di comunicazione che permetta l'accesso indipendente ad ogni banco.
A differenza delle cache multi-livello, questa tipologia di cache non presenta il problema della replicazione dei blocchi, che permette un notevole guadagno di spazio: è possibile così dedurre come, a parità di dimensione, una cache NUCA può contenere più informazioni di una cache UCA o ML-UCA.

1.2.1 S-NUCA Private Channels

In questo modello, ogni banco della cache è connesso al controller con un proprio canale privato, eliminando così ogni possibile problema di conflitto di canale e potendo sfruttare sempre la massima velocità di trasferimento possibile. Tuttavia quello che si presenta è un importante problema di overhead in termini di area di silicio necessaria, comportando così una forte limitazione sul numero di blocchi utilizzabili per la realizzazione della cache.

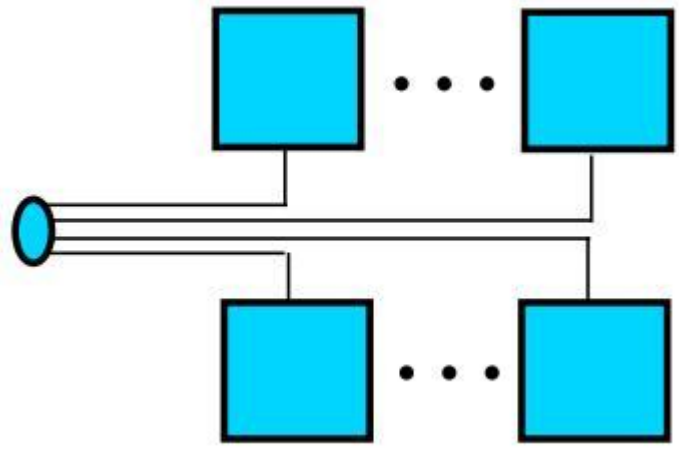


Figura 1.2. Architettutra S-NUCA-1 (Size: 16 MB, 32 banks, Avg. Access time: 34 cycles)

Come mostrato in figura [1.3], ogni banco della cache risulta suddiviso in 4 sottobanchi, gestiti da un pre-decoder centrale, il quale ha il compito di smistare i vari segnali di accesso arrivati al banco.

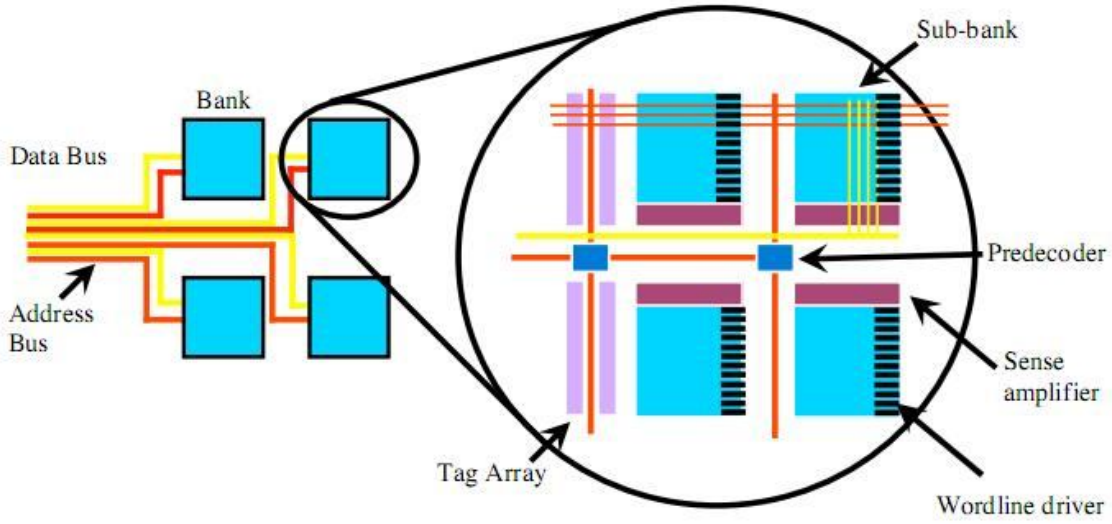


Figura 1.3. Organizzazione interna di una cache S-NUCA-1

La tabella successiva riassume le simulazioni effettuate con memoria S-NUCA realizzate con diverse feature size.

Tech. (nm)	L2 size (MB)	N° banks	Unloaded latency bank	min	max	avg	Conservative Load.	IPC	Aggressive Load.	IPC
130	2	16	3	7	13	10	11.3	0.54	10.0	0.55
100	4	32	3	9	21	15	17.3	0.56	15.3	0.57
70	8	32	5	12	26	19	21.9	0.61	19.3	0.63
50	16	32	8	17	41	29	34.2	0.59	30.2	0.62

Figura 1.4. Performance di una cache S-NUCA-1

In tabella, per ogni dimensione della cache, è riportato il numero di banchi che la costituisce (terza colonna), la latenza di accesso di ogni banco, il tempo di accesso minimo (corrispondente all'accesso al banco più vicino), il tempo di accesso massimo (corrispondente all'accesso al banco più lontano) e il tempo

di accesso medio, cioè la media dei tempi di accesso di ogni banco (dalla quarta alla settima colonna).
Sono inoltre riportati l'IPC e la latenza per i casi di presenza e assenza di conflitti (conservativo e aggressivo rispettivamente).
La principale differenza con i risultati ottenuti per le cache di tipo UCA, consiste nell'aumento dell'IPC al crescere della dimensione della cache e del numero di banchi. Questo avviene poiché all'aumentare della dimensione non c'è un corrispondente aumento della latenza dei banchi delle vie precedenti, le quali rimangono indipendenti le une dalle altre. Tuttavia, questa soluzione risente dell'aumento del numero di collegamenti privati, facendo riscontrare un forte aumento della latenza media e di quella massima nel caso della memoria da 16MB. Questo spiega la riduzione del 2% dell'IPC che subisce quest'ultima memoria, rispetto a quella da 8MB.
E' chiaro dalla terza colonna come la memoria non sia mai composta da più di 32 banchi: questo avviene perché all'aumentare del numero di banchi, i collegamenti crescono in numero e in lunghezza, causando l'aumento della latenza di accesso ai singoli banchi. Non è quindi possibile aumentare la suddivisione in sottobanchi oltre un certo limite, perché questo comporterebbe una perdita di prestazioni, dovuto al maggiore ritardo causato dall'aumento dei collegamenti.

1.2.2 S-NUCA Switched Channels

In questo modello, definito S-NUCA-2, l'elevato numero di fili necessari per la realizzazione dei canali privati del modello precedente, è ridotto grazie all'implementazione di una rete 2-D mesh di collegamenti punto-punto, dove viene utilizzato uno switch per ogni banco. Analisi sperimentali hanno dimostrato che questa architettura offre migliori risultati al crescere della dimensione della cache, ma la complessità introdotta dagli switch la rendono inadatta per cache di piccole e medie dimensioni.

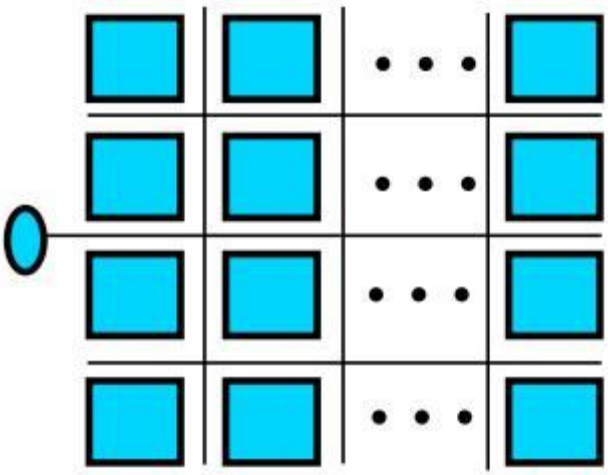


Figura 1.5. Architettura S-NUCA-2 (Size: 16 MB, 32 banks, Avg. Access time: 24 cycles)

La figura [1.5] illustra l'implementazione della rete mesh fra i banchi: in questa configurazione ogni banco ha un proprio switch, che risulta connesso agli switch dei banchi vicini.

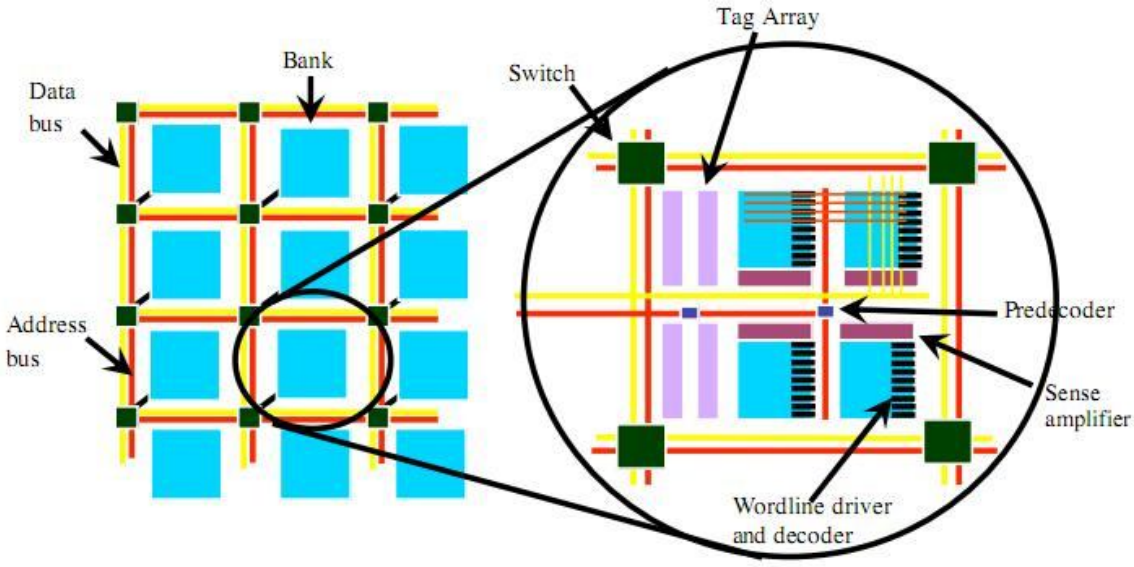


Figura 1.6. Organizzazione interna di una cache S-NUCA-2

La tabella seguente mostra come l'utilizzo della rete mesh permetta una maggiore velocità negli accessi ai banchi. Rispetto all'architettura S-NUCA-1, la latenza dei banchi sia in caso di assenza di conflitti che in presenza di questi, risulta notevolmente ridotta.

Tech. (nm)	L2 size (MB)	Num. banks	Unloaded latency				Loaded latency	IPC	Bank req.
			bank	min	max	avg			
130	2	16	3	4	11	8	9.7	0.55	17M
100	4	32	3	4	15	10	11.9	0.58	16M
70	8	32	5	6	29	18	20.6	0.62	15M
50	16	32	8	9	32	21	24.2	0.65	15M

Figura 1.7. Performance di una cache S-NUCA-2

La latenza del banco coincide con il tempo di accesso a quest'ultimo. I tempi min, max e avg indicano rispettivamente il tempo di accesso ai banchi più vicini, a quelli più lontani e il tempo di accesso medio. Sono inoltre calcolati sperimentalmente l'IPC, la latenza in presenza di carico e il numero di accessi ai banchi.

1.2.3 Memorie cache D-NUCA

A differenza delle memorie di tipo S-NUCA, questi modelli permettono di modificare nel tempo l'associazione dei blocchi con i relativi banchi. Questo meccanismo comporta, come dimostrato da *Kim et al. [1]*, una latenza media minore, concentrando i dati più utilizzati nei banchi in prossimità del controller; questa caratteristica le rende più performanti rispetto alle S-NUCA e risulta fondamentale per l'introduzione di meccanismi per la riduzione del consumo di potenza. La possibilità di avere dati maggiormente riferiti in prossimità del controller permette infatti di ridurre il consumo energetico globale, rispetto ad altre architetture.

Data la loro possibilità di modificare l'associazione blocco-banco, nelle cache D-NUCA viene introdotto, tramite l'implementazione di politiche LRU, il meccanismo di migrazione dei blocchi, necessario a mantenere i dati acceduti più frequentemente nei banchi più prossimi al controller.

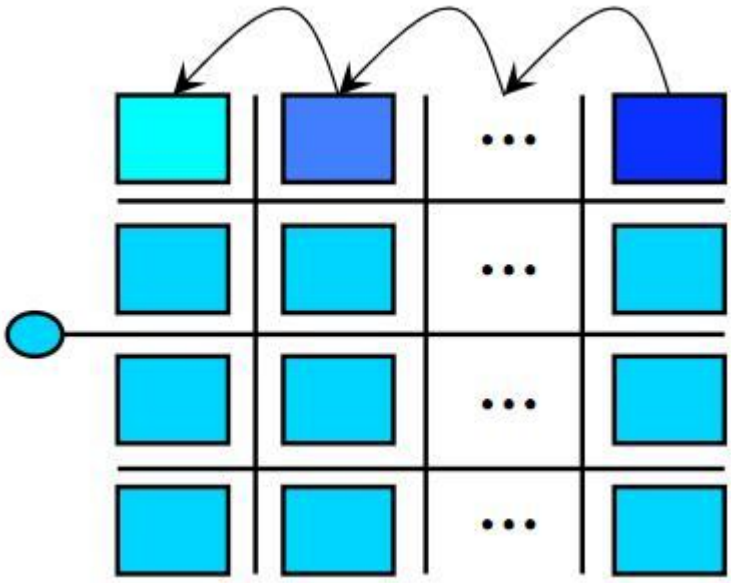


Figura 1.8. Architettura D-NUCA

Per poter implementare questa architettura, è necessario definire le seguenti politiche:

- *mapping*: come vengono associati i blocchi ai rispettivi banchi;
- *search*: come ricercare un blocco all'interno delle sue possibili locazioni;
- *promotion*: sotto quali condizioni i blocchi dovrebbero migrare da un banco all'altro;
- *insertion*: quale blocco rimpiazzare in caso di miss.

1.2.3.1 Mapping dei blocchi

Dato l'alto numero di banchi, è possibile mappare un blocco in un'unica posizione (cache direct-mapped), oppure permettere di mapparlo in modo dinamico in un qualsiasi banco (cache fully-associative). Per le memorie cache di tipo D-NUCA, come compromesso tra flessibilità di piazzamento dei dati e velocità di ricerca, è stato scelto di implementare la soluzione a spread sets, che consiste nel considerare l'intera cache come un'unica struttura set-associative dove ogni set è distribuito su più banchi ed ogni banco contiene una singola via del set.

L'insieme dei banchi usati per realizzare questo tipo di associatività prende il nome di bank set e il numero di banchi del bank set corrisponde all'associatività

della cache. Per assegnare i banchi di una cache ai rispettivi bank set sono state indicate tre strategie: simple mapping, fair mapping e shared mapping.
La strategia simple mapping prevede che ciascuna colonna di banchi rappresenti un bank set: in questo modo un blocco viene individuato cercando per prima la colonna di banchi di appartenenza (cioè il bank set), quindi il relativo set e infine comparando i campi tag delle varie vie con l'indirizzo del blocco in questione. Questa è la politica più semplice da implementare, ma presenta due principali limitazioni: la prima consiste nell'avere l'associatività vincolata all'organizzazione fisica della cache, che può non coincidere con il valore ottimale; la seconda riguarda invece la diversità delle latenze medie dei bank sets, data dalle diverse distanze dei banchi di un set dal controller.

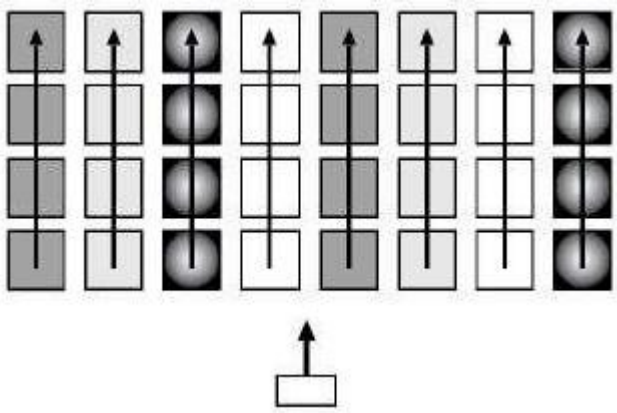


Figura 1.9. Simple mapping

La politica di fair mapping prevede che i banchi siani associati ai bank set in maniera che i tempi medi di accesso dei bank set stessi risultino simili. Lo svantaggio di questa strategie di mapping consiste nella complessità del protocollo di routing necessario alla realizzazione degli spostamenti dei banchi.

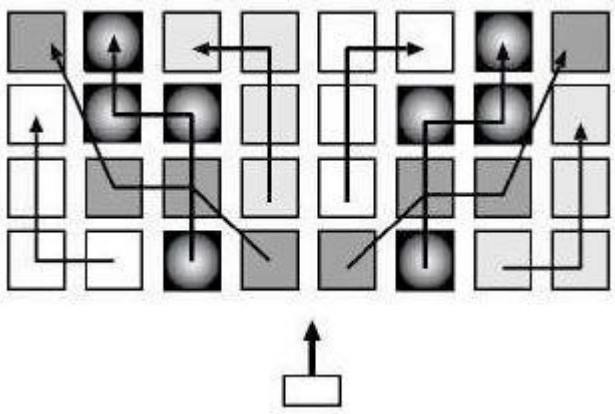


Figura 1.10. Fair mapping

Nella politica shared mapping, i banchi più vicini al controller sono condivisi tra più bank set. Questo permette di offrire ai banchi più usati di ogni insieme la stessa latenza (quella minore possibile). Tutto questo comporta un certo overhead di complessità nella gestione dei banchi stessi.

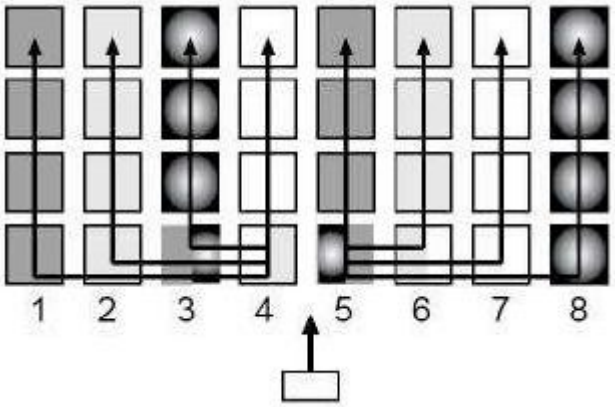


Figura 1.11. Shared mapping

1.2.3.2 Search di un blocco

Dal momento in cui ad un indirizzo di blocco viene associato un determinato bank set, occorre stabilire in quale delle sue vie, e quindi in quale banco, tale blocco è situato. Poichè le cache D-NUCA prevedono un mapping dinamico, questa operazione comporta la ricerca del blocco in ogni via del bank set: è necessario quindi valutare il campo tag di ogni blocco, partendo dal banco più vicino al controller, fino a risolvere l'accesso in una hit o in una miss. Per

effettuare tale ricerca, sono state proposte due modalità principali, chiamate rispettivamente incremental search e multicast search.
La prima metodologia prevede che i banchi componenti il bank set siano acceduti sequenzialmente, partendo da quello più vicino al controller, finchè il blocco non viene identificato o non si verifichi una miss; in questo modo viene ridotto al minimo il traffico sulla rete e viene di conseguenza ridotto il consumo di potenza.
La seconda tecnica permette accessi in parallelo ai bank set, limitando il parallelismo al solo ritardo di routing introdotto dalla rete; l'utilizzo di questo metodo comporta un incremento delle prestazioni nella ricerca del blocco a scapito di un maggior consumo di potenza.
Insieme a queste due modalità sono state sviluppate anche alcune varianti, come la limited multicast e la partitioned multicast, che sono una combinazione delle precedenti.

1.2.3.3 Promotion dei blocchi

Per concentrare gli accessi nei banchi più vicini al controller sarebbe auspicabile avere un ordinamento dei blocchi nei bank set secondo la strategia LRU (Least Recently Used), dove il primo banco di ogni set fosse il blocco MRU (Most Recently Used). Tuttavia, è stato scelto di implementare solamente un'approssimazione di questa metodologia (detta generational promotion), poiché l'esatta implementazione della strategia LRU comporterebbe un eccessivo numero di trasferimenti dei dati e una complessità circuitale non accettabile. Tale strategia prevede che, in caso di hit, un dato blocco venga promosso nella posizione immediatamente adiacente a quella attuale, avvicendandosi con il blocco di cui prenderà il posto, se quest'ultimo è un blocco valido. Nell'architettura di riferimento, non è tuttavia possibile che un banco non sia valido, poiché la cache continua ad essere utilizzata fino a quando non è completamente piena e solo in quel momento entra in gioco il meccanismo di promozione e demozione dei blocchi.

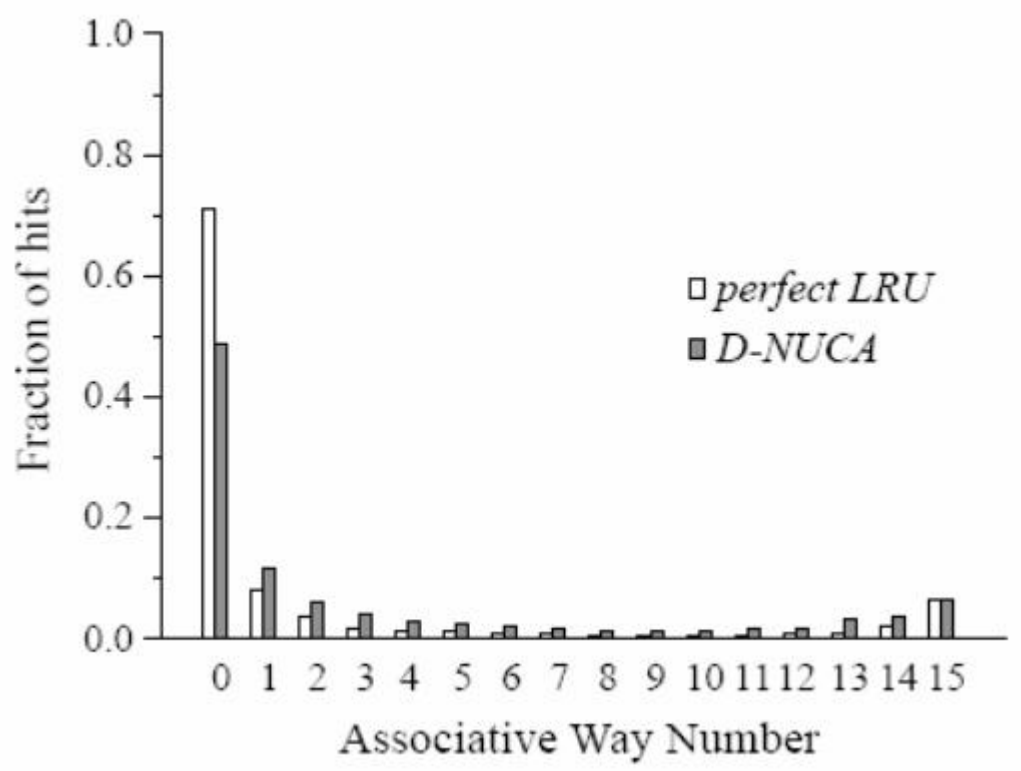


Figura 1.12. Distribuzione delle hit per LRU e generational promotion

In figura 1.12 è mostrata la distribuzione delle hit sui vari set: gli accessi per le cache D-NUCA sono ancora concentrati sui banchi più vicini al controller. Questo comportamento è dovuto al meccanismo di promozione dei blocchi, che permette di ottenere una buona approssimazione di un algoritmo basato su un metodo LRU perfetto.

1.2.3.4 Insertion di un blocco

E' necessario stabilire una politica di inserimento di un blocco in caso di cache miss: *Kim et al.* [1] hanno privilegiato una soluzione che inserisca il nuovo blocco nel banco più lontano tra quelli che costituiscono il bank set scelto per l'operazione. In questo modo si ovvia al problema di rimpiazzare un blocco più vicino al controller, che avrebbe in questo caso più probabilità di essere riacceduto, vista la politica di promotion descritta precedentemente.
Vanno inoltre stabilite le operazioni da effettuare su di un banco che subisce rimpiazzamento: la strategia zero-copy prevede che il blocco venga semplicemente eliminato dalla cache, mentre la strategia one-copy prevede che il blocco da rimpiazzare vada a sostituire un blocco a più bassa priorità. Alla

politica di inserimento in coda è ovviamente associabile solamente la politica zero-copy.

1.2.4 Confronto delle prestazioni

In figura [1.13] sono invece riportate le prestazioni in termini di latenze, IPC e missrate della cache D-NUCA in configurazione base. Nella terza colonna è riportata l'organizzazione della matrice dei banchi per ogni dimensione della cache: la cache da 2MB risulta composta da 4 bank sets ognuno di 4 vie, quella da 4MB ha invece 8 bank sets da 4 vie, quella da 8MB è realizzata da 16 banksets da 8 vie e infine la cache da 16MB è sostituita da 16 banksets da 16 vie ognuno. Nel secondo blocco della tabella si può leggere il tempo di accesso ad ogni banco, la latenza di accesso minima (corrispondente all'accesso al banco più vicino), quella massima (latenza per l'accesso al banco più lontano) e la latenza media di accesso, tutte considerate in assenza di conflitti. Si può notare come a differenza delle cache S-NUCA1 (private channel), le latenze massime sono estremamente basse considerato l'elevato numero di banchi che compone la cache. La cache da 16MB è infatti composta da ben 256 banchi, ma nonostante ciò ha una latenza massima di soli 47 clock cycles (una cache S-NUCA1 della stessa dimensione è composta da soli 32 banchi ed ha una latenza massima di 41 clock cycles).

Questo concede una maggiore libertà nell'aumento del numero di banchi, riducendone le dimensioni e quindi il tempo di accesso introdotto da ognuno di essi. Si nota infatti che il tempo di accesso dei banchi anche per cache di grandi dimensioni è di soli 3 cicli di clock) mentre per soluzioni S-NUCA (di entrambe le tipologie), questo ritardo cresce fino a 8 cicli di clock a causa della dimensione dei singoli banchi.

Per quanto riguarda le prestazioni, si osserva che per le cache di più piccole dimensioni (4MB), si ha un miglioramento del 2% rispetto alle S-NUCA più performanti, mentre per dimensioni maggiori, l'incremento prestazionale arriva fino al 6%.

Il principale svantaggio di questa tipologia di cache riguarda il grande numero di messaggi che viaggiano all'interno della rete di interconnessione e la conseguente crescita del numero di accesso ai banchi (266M contro i 15M delle

cache S-NUCA2 per le cache da 16MB): tale incremento della comunicazione è dovuto anche alla tecnica di ricerca multi cast utilizzata per questa tipologia di cache. Questo fenomeno contribuisce all'aumento del consumo energetico del dispositivo e aumenta la probabilità di conflitti fra due o più accessi contemporanei. I banchi più piccoli e, di conseguenza, il loro numero più elevato disperde i conflitti, permettendo in questo modo di abbassare il missrate e di ottenere un guadagno di IPC.

Tech. (nm)	L2 size (MB)	Bank org.	Unloaded latency				IPC	Miss rate	Bank req.
			bank	min	max	avg			
130	2	4x4	3	4	11	8	0.57	0.23	73M
100	4	8x4	3	4	15	10	0.63	0.19	72M
70	8	16x8	3	4	31	18	0.67	0.15	138M
50	16	16x16	3	3	47	25	0.71	0.11	266M

Figura 1.13. Performance di una cache D-NUCA

Nella tabella seguente sono riportate le prestazioni delle cache NUCA in termini di IPC, rispetto alle memorie tradizionali (UCA), per diverse tecnologie costruttive. Per quanto riguarda le cache NUCA, sono state prese in esame le configurazioni S-NUCA1, S-NUCA2 e D-NUCA. In particolare, i dati delle D-NUCA si riferiscono ad una configurazione che prevede simple mapping, multi cast search, tail insertion e promotion di tipo 1 bank/1 hit.

Type	Technology (nm)	L2 size (MB)	N° banks	IPC
UCA	130	2	1	0.41
UCA	100	4	1	0.39
UCA	70	8	1	0.34
UCA	50	16	1	0.26
S-NUCA-1	130	2	16	0.55
S-NUCA-1	100	4	32	0.57
S-NUCA-1	70	8	32	0.63
S-NUCA-1	50	16	32	0.62
S-NUCA-2	130	2	16	0.55
S-NUCA-2	100	4	32	0.58
S-NUCA-2	70	8	32	0.62
S-NUCA-2	50	16	32	0.65
D-NUCA	130	2	16	0.57
D-NUCA	100	4	32	0.63
D-NUCA	70	8	128	0.67
D-NUCA	50	16	256	0.71

Figura 1.14 Confronto IPC: UCA, S-NUCA-1, S-NUCA-2, D-NUCA

La tabella mostra le previsioni del comportamento delle diverse tecnologie costruttive, in funzione della diminuzione della feature size. A parità di area occupata dal chip, una feature size minore permette di realizzare memorie con un numero più elevato di banchi: le cache D-NUCA si avvantaggiano di questa situazione, permettendo una maggiore scalabilità, grazie alla possibilità di migrare i blocchi con i dati maggiormenti richiesti. E' evidente come la mappatura dinamica dei blocchi, permetta di ottenere un vantaggio prestazionale rispetto alle memorie di tipo S-NUCA-2, che condividono con le D-NUCA la struttura della rete di interconnessione, ma non permettono la migrazione dei dati.

1.3. Tecnica way-adapting

Nell'articolo di *Kobayashi et al.* [3] viene descritto un meccanismo di riconfigurazione dinamica della cache, in grado di ridurre il consumo di potenza statica utilizzando informazioni sulla località dei riferimenti delle applicazioni. La

procedura di riconfigurazione consiste nell'accensione o nello spegnimento delle vie della cache a tempo di esecuzione. Poiché la località dei riferimenti è diversa per ogni applicazione, il contributo che può dare alle performance ogni via di una cache set-associative, non solo varia utilizzando suite di applicazioni differenti, ma varia anche tra le diverse applicazioni di una stessa suite. Tutto ciò si traduce in un ridotto consumo di potenza, a fronte di una accettabile perdita di prestazioni, disabilitando vie della cache a fronte delle esigenze delle singole applicazioni.

Il meccanismo di way-adapting utilizza la tecnologia *Gated-*V_{dd} [6], che introduce nelle celle SRAM un transistor ad alta V_t tra il circuito e un collegamento, quale può essere quello verso l'alimentazione o verso la massa. Il transistor è un elemento di passo, comandato da un segnale di controllo: se il segnale è attivo, questo è in conduzione e la cella SRAM si comporta normalmente, in caso contrario, la cella, essendo isolata dall'alimentazione, per lo stato che aveva memorizzato precedentemente e si porta in una condizione per cui la potenza di leakage è molto bassa. Per limitare l'aumento di area, il transistor di attivazione e il meccanismo di controllo possono essere condivisi da un'intera wordline della SRAM, riducendo in questo modo anche la complessità circuitale.

Questa tecnica comporta però un aumento di miss a causa della perdite di contenuto delle celle, ma è tuttavia possibile introdurre meccanismi che permettano di concentrare i dati più frequentemente acceduti in porzioni ben determinate della cache, così da applicare lo spegnimento a celle che hanno una bassa probabilità di essere nuovamente riferite. Tali meccanismi utilizzano ad esempio la località degli accessi in memoria dei programmi.

In figura 1.15 sono mostrati i risultati relativi alle località dei riferimenti per diverse suite di benchmark: SPECint/fp, MediaBench, anagram e dhrystone. La colonna cache references mostra il numero di accessi in cache L1 instruction e data, ordinati per lo stato del loro campo LRU.

Suite	Benchmark	Functions	Instruction executed	Cache references
SPECint95	go	go program	548M	157M
	gcc	Based on the GNU C Compiler	281M	109M
	compress	UNIX Utility program	80M	28M
	perl	Programming language	21M	8M
SPECfp2000	ammp	Computational Chemistry	21M	8M
	art	Image Recognition / Neural Net.	1B	200M
	equake	Seismic Wave Propagation Simul.	1B	170M
MediaBench	mpeg2encode	Mpeg2 encoder	1B	146M
	mpeg2decode	Mpeg2 decoder	171M	32M
	g721-encode	Voice compression encoder	585M	270M
	g721-decode	Voice compression decoder	558M	256M
	rasta	Speech recognition	19M	6M
Misc	anagram	Puzzle	17M	6M
	dhrystone	Synthetic benchmark	526M	173M

Figura 1.15. Località dei benchmark a confronto

Nella cache way-adaptable la località dei riferimenti viene analizzata sia a livello locale, dove viene considerato lo stato del campo LRU delle entry accedute nell'arco di un certo periodo, sia a livello globale, dove invece viene utilizzata una macchina a stati per regolare il meccanismo di accensione/spegnimento delle vie, determinante soprattutto in situazioni di comportamenti irregolari della località.

Per quanto riguarda il comportamento locale, viene utilizzata una metrica basata sul parametro D (grado di località della cache NUCA), che viene così definito:

$$D=LRU_{count} / MRU_{count}$$

dove LRU_{count} e MRU_{count} rappresentano il numero di riferimenti ai blocchi LRU e MRU dei vari set all'interno di un determinato periodo. Questa metrica non utilizza l'intero stato del campo LRU delle entry riferite per non introdurre eccessiva complessità a livello circuitale.

In figura [1.16 a] e [1.16 b] sono schematizzati due scenari con località differente.

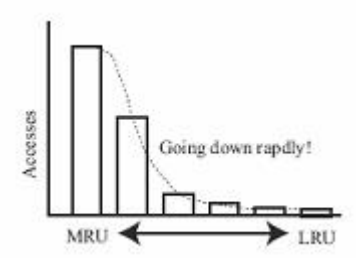


Figura 1.16 a. High Locality Model

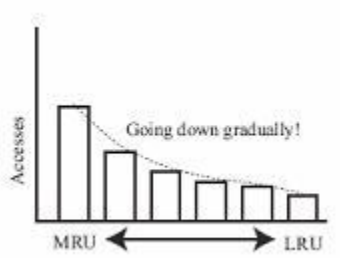


Figura 1.16 b. Low Locality Model

I grafici riportano sull'ascissa gli stati del campo LRU e sull'ordinata il numero di accessi: quando un'applicazione è caratterizzata da un'alta località, la distribuzione degli accessi è concentrata in prossimità dello stato MRU e il numero degli accessi decresce rapidamente procedendo verso lo stato LRU; quando invece la località è bassa, il numero di accessi decresce molto meno velocemente e gli accessi a blocchi LRU non è trascurabile. Con il parametro D è possibile valutare efficacemente il grado di località. Per associare al valore della metrica D un'azione di riconfigurazione, vengono introdotte due soglie, denominate T1 e T2, con il vincolo che T1 sia minore di T2 (in simboli T1 < T2). Rispetto alla coppia di soglie, la metrica D può trovarsi in tre situazioni possibili, determinando così la corrispettiva azione di riconfigurazione (cfr. figura 1.17):

- T1 < D < T2: la configurazione corrente della cache è adeguata alla località dell'applicazione e non è necessaria nessuna riconfigurazione.
- D < T1: il working set dell'applicazione è contenuto pressocchè interamente nella cache e probabilmente ridurre l'associatività della cache non comporterà un degrado delle prestazioni; è possibile quindi spegnere una via o più vie.
- D > T2: il working set dell'applicazione richiede una maggiore dimensione della cache e l'azione da intraprendere è l'espansione della cache.

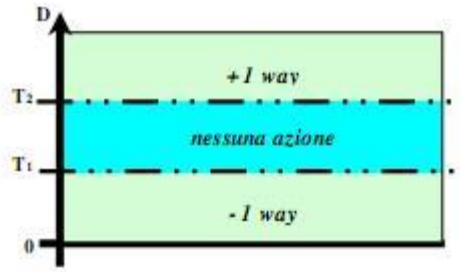


Figura 1.17. Zone di funzionamento del meccanismo di way-adapting

Il comportamento globale dell'applicazione in termini di località è stabilito da una macchina a stati utilizzata per stabilire la consistenza di una richiesta di riconfigurazione che si presenta ad un dato istante, basandosi sui precedenti valori assunti dal parametro D. Alla fine di ogni periodo, se si presenta la necessità di effettuare una riconfigurazione, questa viene effettivamente eseguita solo se lo stato ottenuto con l'utilizzo della macchina a stati è coerente.

Sono previsti tre tipi di stato: incremento di una via (+1), mantenimento della configurazione corrente (0) e decremento di una via (-1). Se la macchina a stati viene realizzata utilizzando n bit per codificare gli stati, sono necessarie almeno n^2 - 1 transazioni per passare dallo stato +1 a quello -1. In figura [1.18 a] e [1.18 b] è riportato il diagramma della transizioni della macchina a stati, in versione simmetrica e asimmetrica: la versione asimmetrica è indicata per ottenere maggiori prestazioni a scapito di un aumento del consumo di potenza, perché il suo comportamento privilegia le richieste di accensione delle vie rispetto a quelle di spegnimento.

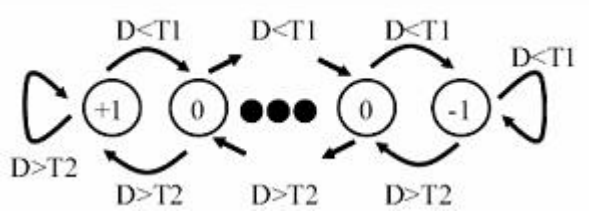


Figura 1.18 a. Diagramma delle transizioni della macchina a stati simmetrica

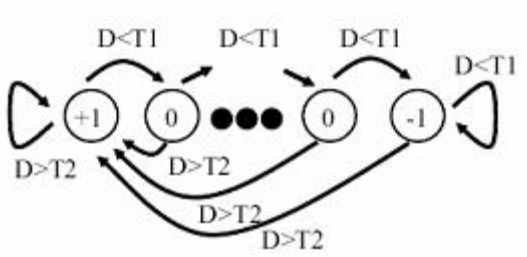


Figura 1.18 b. Diagramma delle transizioni della macchina a stati asimmetrica

Nel momento in cui il meccanismo determina lo spegnimento di una via, tale via viene scelta in maniera casuale tra quelle componenti la cache, poichè l'analisi che è stata condotta da Kobayashi et al [3] evidenzia come l'utilizzo di politiche alternative per la selezione delle vie da spegnere non determini significativi aumenti prestazionali. Tale meccanismo causa ovviamente miss aggiuntive dovute al riferimento di blocchi che si trovavano nella via disattivata, ma l'utilizzo di cache D-NUCA può compensare questo comportamento con il meccanismo di migrazione dei blocchi.

Il comportamento della tecnica del way-adapting ha il vantaggio di non dipendere da parametri specifici di una singola applicazione o di una determinata tipologia di applicazioni, rimanendo però ugualmente flessibile anche relativamente alla configurazione della cache cui si voglia applicare. Tale flessibilità del way-adapting è dovuta in gran parte all'adozione della metrica D, che permette di identificare efficacemente il grado di località dei programmi senza utilizzare parametri assoluti (come ad esempio il miss-rate), che variano a seconda dell'applicazione considerata. Tuttavia questa tecnica è maggiormente efficace quando utilizzata con cache ad elevata associatività, poichè se ne può derivare un maggiore risparmio energetico.

Per poter implementare il meccanismo di way-adapting, occorre selezionare opportunamente le soglie T1 e T2 e, relativamente alla macchina a stati, la tipologia (simmetrica o asimmetrica) e il numero di bit per la codifica degli stati. In particolare, per quanto riguarda le soglie, nella maggior parte delle simulazioni realizzate nel lavoro di Kobayashi, sono stati scelti come coppia [T1; T2] i valori 0.005 e 0.02, per cui sono ottenute le migliori performance. Nell'utilizzo della macchina a stati, ha dimostrato più stabilità l'implementazione a 3 bit rispetto a quella a 2 bit, mentre l'impiego di un numero maggiore di bit ha dimostrato un comportamento troppo conservativo. Inoltre, la versione

asimmetrica ha confermato prestazioni migliori rispetto a quella simmetrica, risultando però quest'ultima più conservativa e offrendo un maggiore risparmio energetico.

1.3.1 Definizione di una cache D-NUCA way-adaptable

Foglia et al, hanno esteso la tecnica del way-adapting alle cache D-NUCA: il meccanismo di migrazione dei blocchi permette di mantenere il working set dell'applicazione all'interno dei banchi (e quindi delle vie) più prossime al controller, rendendo questo tipo di memorie particolarmente adatte all'implementazione della tecnica del way-adapting, relegando le vie da spegnere a quelle più remote, dove si troveranno, per definizione stessa di cache D-NUCA, i blocchi meno riferiti e pertanto più adatti ad essere rimossi [9, 10, 11, 12]. Il comportamento intrinseco di queste memorie risulta inoltre utile nella riduzione del fenomeno di aumento delle miss, dovuto allo spegnimento di una via della cache; questo perchè, a differenza della cache tradizionale in cui la via da spegnere è scelta a caso, nelle D-NUCA la scelta ricade sempre in quella con i blocchi meno riferiti e cioè quella più lontana dal controller, attualmente attiva. Ovviamente, va tenuto conto che tale vantaggio è legato all'efficienza dell'algoritmo di promozione implementato nella cache D-NUCA considerata.

Dato che in questa tipologia di cache l'ordinamento LRU è solamente approssimato, è necessario definire una metrica per rappresentare lo stato LRU dei blocchi che vengono riferiti. E' necessario infatti distinguere tra accessi globali, ovvero l'accesso all'intera cache, che si ha quando un indirizzo fisico perviene al controller della D-NUCA, e accessi locali, che sono quegli accessi effettuati ad ogni singolo banco, durante la fase di ricerca del blocco riferito.

Dall'analisi delle hit e degli accessi locali per una determinata serie di benchmark [6], si è evidenziato come la località del programma sia più correttamente evidenziata dal numero di hit, poiché negli accessi entrano in gioco fattori legati prevalentemente alle politiche di implementazione della D-NUCA. Pertanto il meccanismo del way-adapting deve sfruttare una metrica basata sull'utilizzo delle hit. In particolare, il parametro D, può essere esteso a questa tipologia di cache sostituendo MRU_{count} con il numero di hit sulla via più

vicina al controller e LRU_{count} con il numero di hit sulla via più remota tra quelle attive nell'istante in cui viene valutata la metrica. In formula:

D = (hit on the farthest way count) / (hit on the nearest way count)

L'algoritmo way adapting prevede la valutazione di questa metrica periodicamente: la frequenza con cui la metrica D viene valutata è definita pollrate e determinata dal parametro K. Pertanto, la tecnica del way-adapting si avvale del seguente algoritmo:

```
each K hit {
     evaluate D
     choose the right action using [T1;T2]
}
```

Nel lavoro di tesi di Di Mari [7], è stata introdotta una nuova metrica, chiamata D differenziale normalizzata e definita come:

$$D' = (D_i - D_{i-1}) / D_i$$

dove D è il rapporto tra il numero di hit sull'ultima via (quella più lontana dal controllore L2) e il numero di hit sulla prima via (quella più vicina al controllore L2).
Per uno studio più approfondito si rimanda ai lavori di tesi svolti da Gabrielli [6] e Di Mari [7].

2. Esplorazione dell'albero delle riconfigurazioni

Al fine di poter applicare la tecnica del way-adapting, è necessario identificare una coppia di valori T1 e T2, che costituiranno le soglie di riferimento per il discernimento dell'operazione di riconfigurazione da effettuare.
L'identificazione delle soglie è quindi un punto fondamentale dell'intero procedimento di implementazione dell'algoritmo, poichè questo sarà tanto più efficace ed efficiente quanto le soglie scelte saranno accurate. Per ottenere la coppia di valori [T1; T2] è necessario compiere due passi fondamentali:

1. Esplorare l'albero delle riconfigurazioni per una determinata applicazione campione;
2. Isolare una coppia di valori [T1;T2] ottimali.

L'albero delle riconfigurazioni al passo 1, è ottenuto attraverso una procedura di esplorazione offline delle possibili riconfigurazioni, per cui ad ogni passo, viene scelta la riconfigurazione che totalizza il miglior comportamento a fronte della valutazione di una determinata metrica. Questa procedura è inoltre non predittiva, in quanto per poter scegliere il comportamento ottimale è necessario conoscere tale comportamento al passo successivo e solo allora scegliere come proseguire.

Le metodologie per la determinazione delle soglie verranno trattate nel capitolo successivo, mentre ci soffermeremo adesso sulle tecniche di esplorazione dell'albero delle riconfigurazioni vagliate in questo lavoro di tesi.

2.1 Considerazioni sull'utilizzo di una fase di warmup

Prima di passare alle tecniche di esplorazione vere e proprie, è opportuno discutere sull'utilizzo o meno di una fase di warmup della cache, prima dell'esecuzione della fase di simulazione vera a propria, definendo innanzitutto cosa si vuole intendere con il termine warmup.
All'avvio dell'applicazione infatti, la cache si trova in una situazione in cui nessun dato relativo a questa è ancora stato inserito (cache scarica), pertanto i primi accessi alla cache genereranno sicuramente miss (cold misses).

L'utilizzo di una fase di warmup consente di analizzare il comportamento dell'applicazione allo stato di regime, evitando così di inficiare le performance inserendo delle miss inevitabili. Da un punto di vista non puramente simulativo, tenere conto delle cold misses, rappresenta una migliore approssimazione del comportamento reale della macchina simulata.
In questa parte del lavoro di tesi, si è scelto di eseguire le esplorazioni dell'albero delle riconfigurazioni in entrambi i modi, allo scopo di identificare le effettive differenze prestazionali riscontrabili.

2.2 Realizzazione della fase di warmup

Le simulazioni eseguite durante il lavoro di tesi sono effettuate, in accordo con [1], eseguendo, per ogni benchmark, la sequenza di istruzioni che racchiude la fase di maggiore attività della singola applicazione. La fase di fast forwarding (FFWD) rappresenta il numero di istruzioni saltate per raggiungere la fase RUN, che rappresenta il numero di istruzioni effettivamente simulate.
Per poter confrontare i risultati delle simulazioni effettuate con e senza l'utilizzo del warmup, è necessario che la fase eseguita dell'applicazione interessata sia la stessa per entrambe le simulazioni. Per realizzare questa caratteristica, il warmup viene inserito tra la fase di FFWD e quella di RUN, in modo che $FFWD_{warmup}$ si pari a FFWD senza warmup meno il numero di istruzioni della fase di warmup; in formula:

$$FFWD_{warmup} = FFWD - WARMUP$$

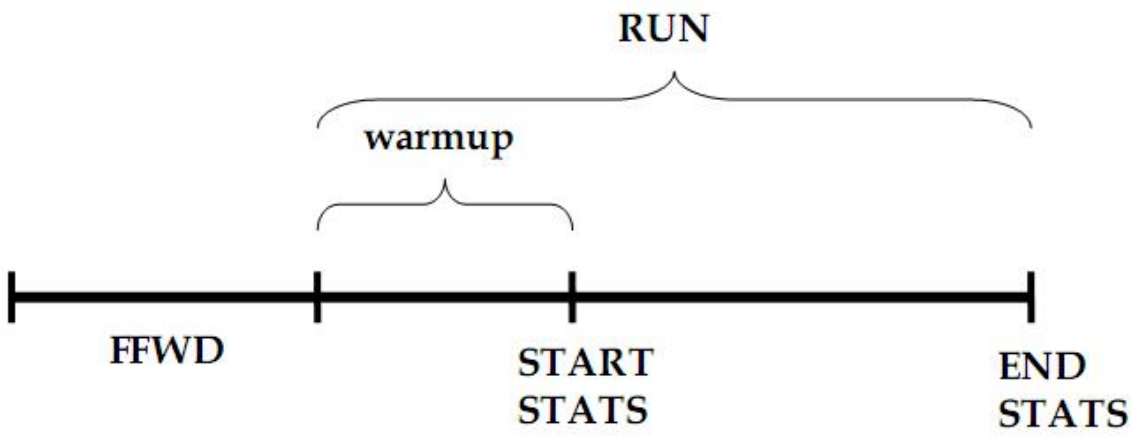


Figura 2.1. relazione tra le diverse fasi della simulazione

dove FFWD, RUN, WARMUP e $FFWD_{warmup}$ sono rispettivamente il numero di istruzioni da eseguire nella fase di fast forwarding, per la fase di run, per quella di warmup e per quella di fast forwarding relativa al caso in cui venga utilizzata la fase di warmup.

In questo modo, come le fasi di RUN di entrambe le simulazioni coincidono e i risultati così ottenuti posso essere confrontati.

In accordo con il lavoro di tesi di Di Mari [7], il valore scelto per la dimensione della fase di warmup è stimato pari a 50 milioni di istruzioni; per questo valore infatti le performance ottenute dalla simulazioni saranno prossime a quelle che si otterrebbero simulando completamente l'intera fase del benchmark.

2.3 Metodologia generale di esplorazione (tuning)

Come è stato detto precedentemente, al fine di poter estrarre una coppia di soglie utili all'algoritmo del way-adapting, è necessario esplorare l'albero delle riconfigurazioni per un dato benchmark, secondo una metrica alternativa a D. La metodologia di esplorazione consiste nello scegliere, ad ogni fase, l'azione di riconfigurazione più conservativa all'interno di una data soglia di ottimalità: questo è possibile perché la metodologia di esplorazione è non predittiva, ovvero è possibile decidere quale azione intraprendere conoscendone le conseguenze. Nel corso di questo capitolo ne verrà valutata l'applicazione per ogni metrica utilizzata e l'utilizzo di soglie di ottimalità diverse, in base alla frequenza di polling dell'algoritmo.

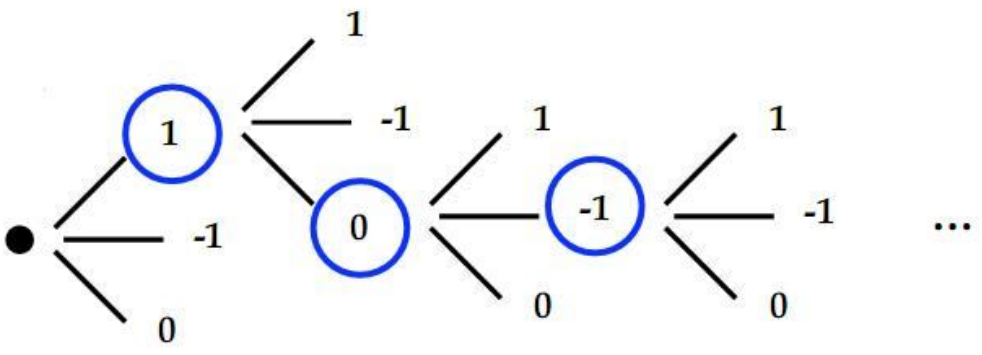


Figura 2.2. Esempio di costruzione dell'albero delle riconfigurazioni (i rami evidenziati corrispondono alla scelta fatta)

La scelta dei rami da seguire durante la simulazione porterà alla costruzione di un cammino sub-ottimo, ovvero non realizzerà un cammino ottimale, ma bensì si otterà una buona approssimazione di tale cammino.

2.3.1 Lower global miss-rate (gmr)

In questa implementazione della politica di esplorazione, il ramo vincente dell'attuale step di configurazione viene scelto ricercando applicando la metodologia decisionale sulla metrica global missrate. Tale metrica considera il valore del missrate raggiunto da ogni singolo ramo a partire dall'inizio della fase RUN della simulazione fino al momento in cui la metrica viene valutata, portando quindi con sé una componente di memoria della storia passata della simulazione. In questa realizzazione, la decisione del ramo che verrà utilizzato per proseguire la simulazione è presa in accordo alla seguente formula:

$$missrate[winner] \leq min\_missrate*(100 + threshold)/100;\ dove:\ threshold = 1$$

Il parametro threshold, rappresenta una soglia di tolleranza entro la quale poter scegliere una diversa azione di riconfigurazione, se questa è più conservativa di quella corrispondente all'azione con il minor missrate. Durante il lavoro di tesi è stato osservato come, al variare del K, questo parametro debba essere a sua volta variato: questo perché una frequenza di polling più rapida, permette di accumulare meno informazioni, riducendo le grandezze in gioco. Pertando mantenere fisso il parametro threshold, comporterebbe creare range di tolleranza troppo ampi al decrescere di K: è stato osservato sperimentalmente come scalare linearmente la tolleranza con il pollrate, mantenga efficiente la tecnica decisionale.

Se il programma è soggetto a molte miss, le prestazioni sono ulteriormente penalizzate dal fatto che la risoluzione in miss di un accesso alla cache si determina solo quando sono stati acceduti tutti i banchi del bankset in questione

e il messaggio di notifica della miss ha attraversato la rete in senso opposto alla richiesta, giungendo così al controller.

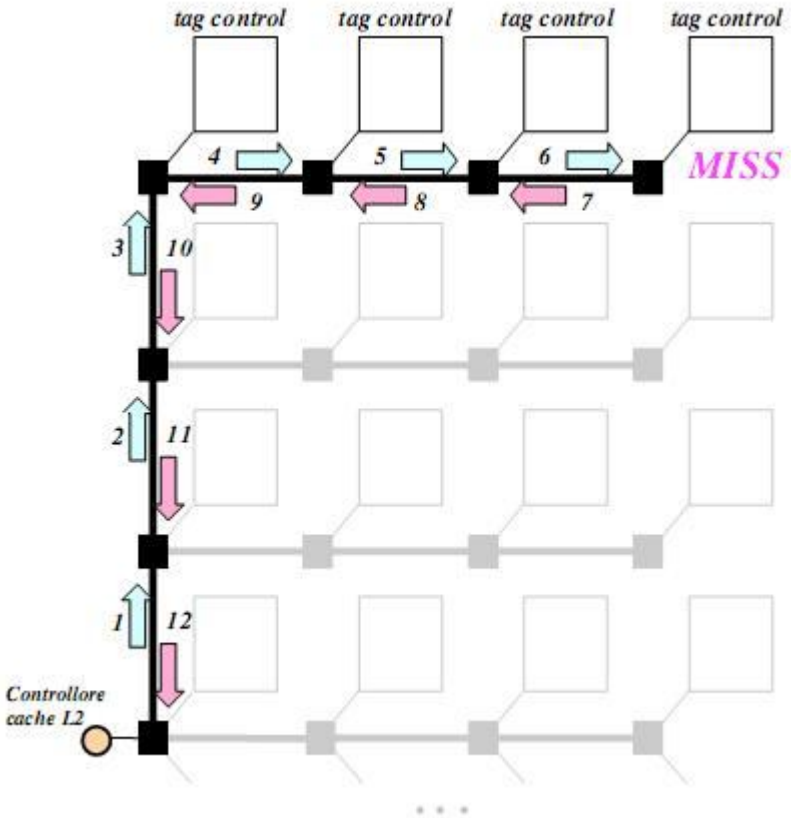


Figura 2.3. Stati successivi in caso di miss

Per chiarire questo aspetto, in figura [2.3] è riportato lo schema degli eventi che si succedono nel caso di una miss. Le frecce azzurre indicano il percorso seguito dal pacchetto di richiesta, mentre quelle rosa indicano il percorso del pacchetto di notifica di miss. La dicitura “tag control” indica che in corrispondenza di quel banco l’accesso è stato interrotto subito dopo il controllo del campo tag. Quando viene effettuato l’accesso ad un banco, la presenza o meno del blocco richiesto è già determinabile subito dopo il controllo del campo tag e, in caso di miss, l’intero accesso può essere abortito anticipatamente, riducendo la latenza complessiva e il consumo di potenza.

2.3.2 Lower local miss-rate (lmr)

Come variante alla metrica precedente, è stato scelto di calcolare il miss-rate localmente ad ogni riconfigurazione. In questo modo viene eliminato l’effetto

memoria contenuto nel metodo di calcolo usato precedentemente, riuscendo ad apprezzare maggiormente le variazioni istantanee del miss-rate tra due riconfigurazioni consecutive.

2.3.3 Higher global IPC (gipc)

In alternativa al miss-rate, è possibile utilizzare come metrica di riferimento per l'esplorazione dell'albero, il valore di IPC (instruction per cycle). Essendo un indice di performance, è necessario questa volta considerare il massimo valore ottenibile, pertanto la decisione del vincitore sarà presa secondo la seguente formula:

$$ipc[winner] \geq max\_ipc*(100 - threshold)/100; \text{ con } threshold = 1$$

Anche in questo caso, come per la metrica global miss-rate, il valore dell'IPC calcolato conterrà una componente di memoria dell'intera simulazione...

2.3.4 Higher local IPC (lipc)

Anche per la metrica IPC è stato scelto di eseguirne il calcolo localmente, in modo da avere un prospetto completo del comportamento di queste tecniche. Analogamente al caso precedente, il ramo vincente di ogni passo della simulazione è scelto in base all'azione più conservativa all'interno della soglia di tolleranza.

2.4 Variazione dei parametri di tuning

L'utilizzo del tuning consente di stabilire il cammino da seguire lungo l'albero delle riconfigurazioni tale per cui è possibile ottenere i risultati ottimali per una data metrica di esplorazione e un dato pollrate. Pertanto, utilizzando il tuning, è possibile stabile quale sia la metrica che ottiene il comportamento migliore ad un dato pollrate e quale sia invece il miglior pollrate utilizzabile con una data metrica.

Viste queste considerazioni, è stata strutturata una metodologia di indagine atta a identificare il pollrate ottimale da utilizzare nell'algoritmo way-adapting:

- esecuzione del tuning con differenti valori di pollrate (K) per la metrica utilizzata abitualmente nel tuning (minor missrate globale)
- esecuzione del tuning con le differenti metriche di esplorazione utilizzando il pollrate che ha ottenuto il miglior comportamento al passo precedente
- riesecuzione del tuning con differenti valori di K per la metrica di esplorazione vincente

Con il primo passo si confrontano i comportamenti della metrica di esplorazione utilizzata per il tuning nel precedente lavoro di tesi [7], per differenti valori di pollrate, valutando il K che ottiene il miglior trade-off. A questo punto, il passo successivo tende a valutare il comportamento del pollrate identificato, con metriche di esplorazioni differenti. In questo modo si valuta la bontà del parametro K in relazione alle diverse metriche. La metrica con il miglior comportamento viene poi confrontata con valori differenti di pollrate. In questo modo, è possibile esplorare il comportamento del pollrate in relazione alla sua variazione e a alla variazione delle metriche di esplorazione, permettendo di stabilire la bontà di un determinato valore.

3. Tecniche alternative di calcolo delle soglie

In questo capitolo verranno trattate le tecniche di identificazione delle soglie necessarie al funzionamento dell'algoritmo way-adapting. Il primo metodo affrontato è quello utilizzando comunemente e consiste nella risoluzione di un sistema di 2*n disquazioni di primo grado, dove le incognite sono le soglie.
I due metodi proposti successivamente affrontano il problema dell'identificazione delle soglie massimizzando il numero di vincoli risolti. Il secondo metodo inoltre, avvalendosi dell'utilizzo di grafici, consente di avere rapidamente un quadro della distribuzione dei vincoli nello spazio delle soluzioni.

3.1 Metodo delle disequazioni

La tecnica del tuning permette di svincolare la scelta dell'azione di riconfigurazione dalle soglie e dalla metrica D, forzandola in base alla metrica di esplorazione utilizzata per l'esplorazione. Con questa tecnica di estrazione delle soglie, T1 e T2 vengono considerate come le incognite di un sistema di disequazioni ottenute considerando ad ogni riconfigurazione i seguenti vincoli:

if (action == +1) then T2 < Dckpt[i] && T1 < Dckpt[i]
if (action == 0) then T1 < Dckpt[i] < T2
if (action == -1) then Dckpt[i] < T1 && Dckpt[i] < T2
T1 < T2

In questo caso la metrica D è nota perché viene calcolata ad ogni checkpoint.
Risolvere un sistema di disequazioni significa trovare le regioni di spazio che soddisfano l'intero sistema; può accadere però che sia impossibile soddisfare tutti i vincoli del sistema, ovvero che il sistema sia irrisolvibile e che non abbia quindi un'unica soluzione reale. In questo caso, è necessario scegliere quale regione di piano adottare per le due incognite T1 e T2, andando a vedere le porzioni di piano con più intersezioni derivanti dalle disequazioni in esame. La politica adottata in questa tecnica per la selezione dei vincoli da rilasciare consiste nell'attribuire un peso ad ogni disequazione, decrescente con

l'aumentare del numero d'ordine del checkpoint a cui tale vincolo è stato generato. Questo perché azioni avvenute all'inizio della fase di RUN sono cruciali nell'economia delle prestazioni finali, determinando maggiormente la serie delle riconfigurazioni. Sarebbe infatti più difficile, a tempo di esecuzione, recuperare un errore compiuto nei primi istanti di esecuzione, poiché tale errore comporterebbe un allontanamento significativo dal percorso ottimale.
Inoltre, è necessario risolvere la scelta della regione di piano da utilizzare, considerando che tale metologia consente di ottenere soluzioni multiple del sistema: tale decisione viene presa tenendo un contatore che indica il numero di vincoli risolti da una determinata regione di spazio, optando per quella che risolve il maggior numero di vincoli.

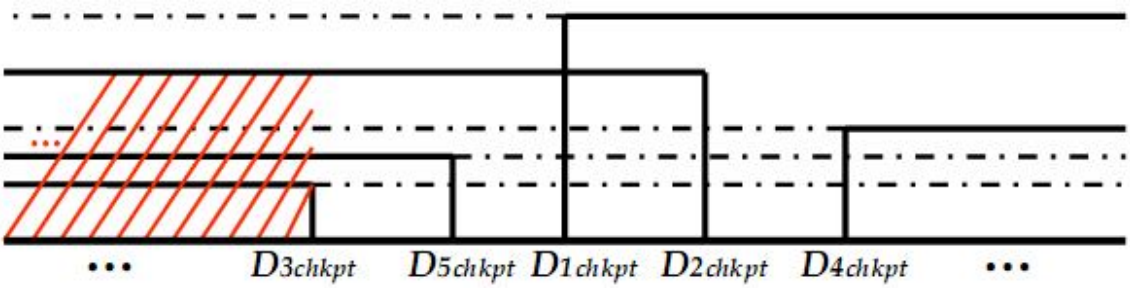
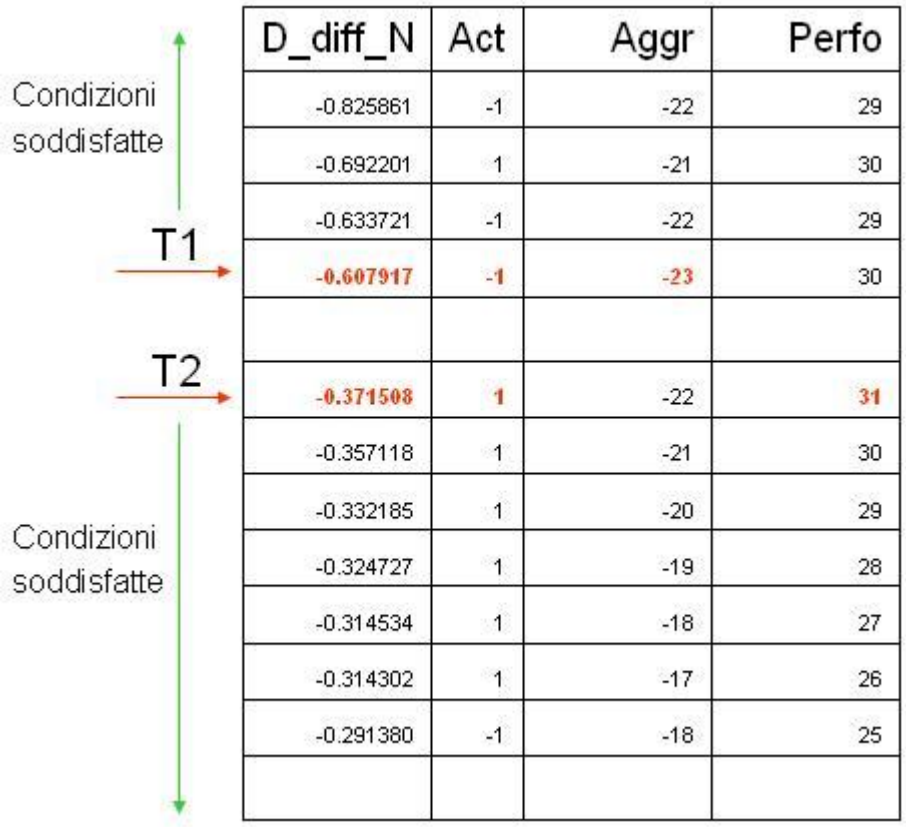


Figura 3.1. Soluzione grafica di un insieme di disequazioni

In figura [3.1] è mostrato un esempio di soluzione del sistema, dove la regione di spazio in rosso rappresenta quella con il maggior numero di vincoli risolti e, pertanto, quella vincente.

3.2 Metodo del conteggio

L'obbiettivo di questo metodo è massimizzare il numero di vincoli soddisfatti dalle soglie T1 e T2. Nel caso della soglia T1, un vincolo è soddisfatto se, dato il valore di D per quel vincolo, questo è minore di T1; dualmente, per la soglia T2, un vincolo è soddisfatto se il valore della metrica D è maggiore di tale soglia. A questo punto, per trovare il massimo numero di vincoli soddisfatti da T1 e T2, basterà ordinare in ordine crescente i valori delle metriche e contare le azioni di riconfigurazione.



D_diff_N	Act	Aggr	Perfo
-0.825861	-1	-22	29
-0.692201	1	-21	30
-0.633721	-1	-22	29
-0.607917	**-1**	**-23**	30
-0.371508	**1**	-22	**31**
-0.357118	1	-21	30
-0.332185	1	-20	29
-0.324727	1	-19	28
-0.314534	1	-18	27
-0.314302	1	-17	26
-0.291380	-1	-18	25

Figura 3.2. Esempio di identificazione delle soglie con il metodo del conteggio

La figura [3.2] mostra come vengono estratte le soglie attraverso il conteggio dei vincoli soddisfatti: la colonna D_diff_N indica la metrica D utilizzata (in questo caso differenziale normalizzata), la colonna act mostra l'azione corrispondente, mentre le colonne Aggr e Perfo indicano rispettivamente il numero di vincoli soddisfatti dal corrispondente valore della metrica D se utilizzata come soglia T1 e il numero di vincoli soddisfatti nel caso in cui tale valore sia invece utilizzato come soglia T2. E' banale osservare quindi come il minimo valore negativo nella terza colonna indichi il valore di D corrispondente come un buon candidato come soglia T1 e il massimo valore positivo nella quarta colonna indichi il corrispondente valore di D come un buon valore per T2.

E' importante notare come sia inutile in questo metodo considerare le rinconfigurazioni che effettuano un mantenimento, poiché non introdurrebbero un contributo significativo. La mancata considerazione dei vincoli che indicano mantenimento è anche il principale punto a sfavore di questo metodo, poiché (come è facile notare anche dalla figura precedente) si tende a determinare

soglie loro ravvicinate che tenderanno a fare oscillare la cache, impedendo all'algoritmo di inseguire l'attività del benchmark. E' possibile vedere un esempio di utilizzo di questo metodo più avanti nel testo (cfr. utilizzo del metodo del conteggio).

3.3 Metodo del clustering

Attraverso questo metodo è possibile visualizzare graficamente la distribuzione delle riconfigurazioni, permettendo di identificare le zone con la più alta concentrazioni di riconfigurazioni.

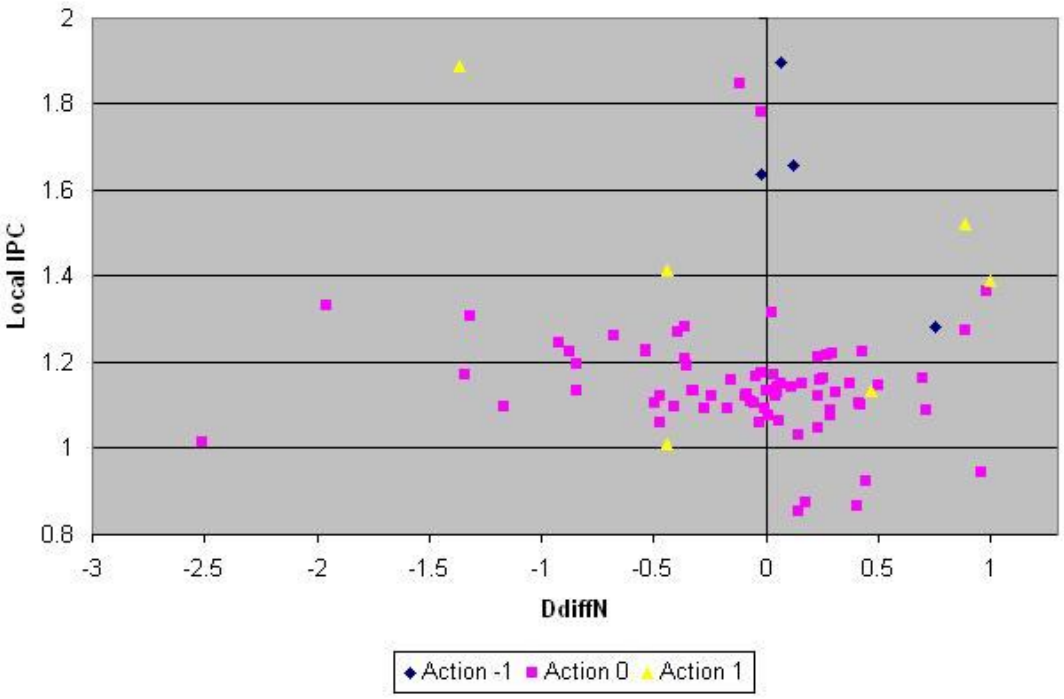


Figura 3.3 a. Grafico della distribuzione delle riconfigurazioni

La figura [3.3 a] mostra la distribuzione delle riconfigurazioni nello spazio, mostrando come non sia possibile tenere conto dell'evoluzione temporale delle riconfigurazioni. Attraverso questo grafico è possibile identificare una prima coppia di soglie, in base al numero e al tipo di vincoli che si vogliono soddisfare.

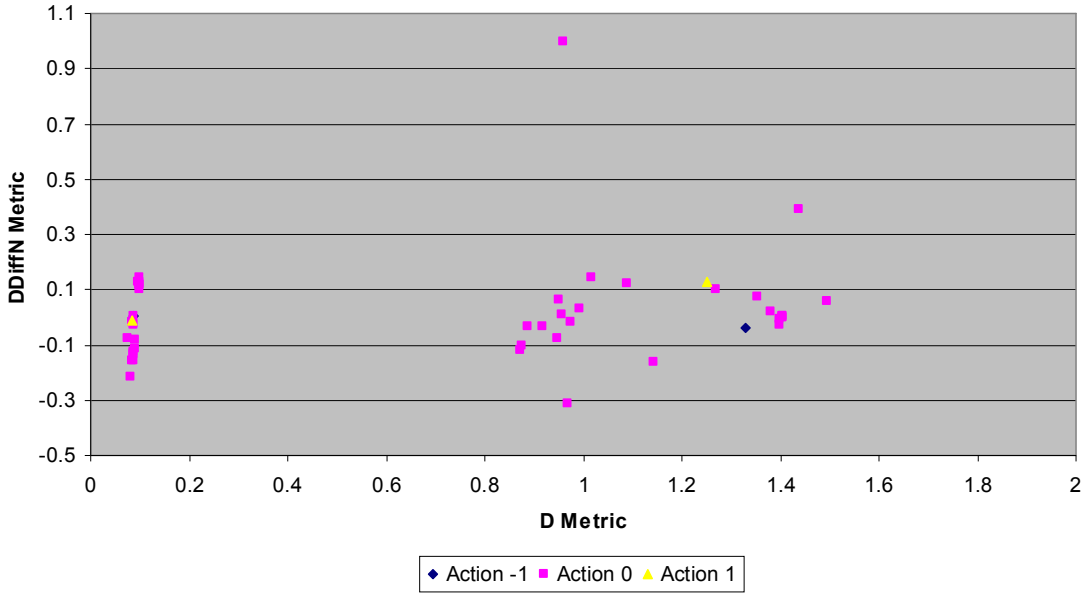


Figura 3.3 b. grafico della distribuzione delle riconfigurazioni

In figura [3.3 b], è mostrato il grafico della distribuzione delle riconfigurazioni per il benchmark bt e K = 100K hit. Le grandezze interessate sono la metrica D e metrica D differenziale normalizzata [7]. E' importante notare come questo grafico dimostra che l'utilizzo della metrica D differenziale normalizza permetta di ottenere vincoli meno sparsi.

Questo metodo può essere considerato complementare al metodo del conteggio, permettendo di tarare le soglie in modo da soddisfare anche un elevato numero di vincoli di mantenimento. E' possibile vedere un esempio di utilizzo di questo metodo più avanti nel testo (cfr. utilizzo del metodo del clustering).

4. Strumenti utilizzati

In questo capitolo verranno descritti gli strumenti utilizzati per effettuare il lavoro di tesi. Dovendo infatti modificare frequentemente i parametri di funzionamento dell'algoritmo way-adapting, sono state realizzate versioni modificate dei simulatori tuning e way-adaptable, in grado di accettare in ingresso i parametri necessari a selezionare metodologie diverse di esplorazione e tecniche diverse di calcolo della metrica. Per il simulatore tuning è stato inoltre ridotto il tempo di calcolo necessario a eseguire una singola simulazione, adottando una tecnica di programmazione multi processo, che permette di valutare in parallelo tutti i rami di una singola riconfigurazione e di mantenere memoria dello stato della simulazione, permettendo di proseguire dal punto in cui questa è stata sospesa per valutare il ramo da seguire.

4.1 Il simulatore Sim-alpha

Per l'esecuzione delle simulazioni necessarie allo svolgimento di questo lavoro da tesi, sono stati utilizzati due simulatori, tutti derivati dal software sim-alpha [9]. Sim-alpha è un simulatore di tipo execution driven, in grado cioè di simulare il comportamento dei singoli stati della pipeline di un microprocessore.

Il microprocessore su cui è basato è l'Alpha 21264, una cpu a 64 bit superscalare con esecuzione di tipo out of order, destinata al mercato dei server ad alte prestazioni. I risultati delle simulazioni effettuate su sim-alpha sono molto accurati poiché il simulatore è stato validato rispetto ad hardware reale.

Le versioni utilizzate in questo lavoro sono sim-alpha tuning multi-processo e sim-apha way adapting, quest'ultimo modificato per l'utilizzo della tecnica differenziale normalizzata introdotta nel lavoro di tesi [7]. Il primo invece è un'evoluzione del simulatore sim-alpha way-halting, modificato per poter eseguire l'intera simulazione tuning e per lavorare con una tenica di programmazione multi-processo.

Di seguito verranno descritte le modifiche apportate da questi simulatori.

4.2 Simulatore multi-processo per il tuning

Come detto precedentemente, per poter estrarre una coppia di soglie, è necessario eseguire l'esplorazione dell'albero delle riconfigurazioni, utilizzando una certa metrica di esplorazione.
Questo tipo di esplorazione, comporta la necessità di eseguire ad ogni passo tre diverse simulazioni (una per ogni operazione possibile) e successivamente sceglierne la migliore secondo la metodologia utilizzata. La precedente versione del simulatore tuning necessitava di rieseguire tutto il cammino precedente prima di poter valutare il nuovo ramo, richiedendo un tempo di calcolo proporzionale alla somma dei primi n numeri, dove n è la profondità dell'albero.
Dovendo valutare valori di pollrate differenti, la profondità dell'albero per ogni simulazione cresce come il prodotto tra i due valori di pollrate e la profondità dell'albero di riferimento.
E' evidente come mano a mano che il pollrate dimunisce il tempo di calcolo per una simulazione con tale valore di K cresca più che linearmente.

Per ovviare a questo problema, è stata realizzata una versione del simulatore in grado di parallelizzare il calcolo dei differenti rami e di far continuare la simulazione solamente al ramo vincente. In questo modo il tempo di simulazione cresce linearmente con il numero di decisioni da intraprendere. In questa versione, ad ogni passo, il processo che è attualmente in esecuzione genera tre sottoprocessi (uno per ogni possibile azione di riconfigurazione) e si sospende in attesa che questi eseguano il ramo dell'albero assegnatogli; al suo risveglio, lascia il controllo della simulazione al processo che ha totalizzato la metrica ottimale, terminando se stesso e gli altri processi.

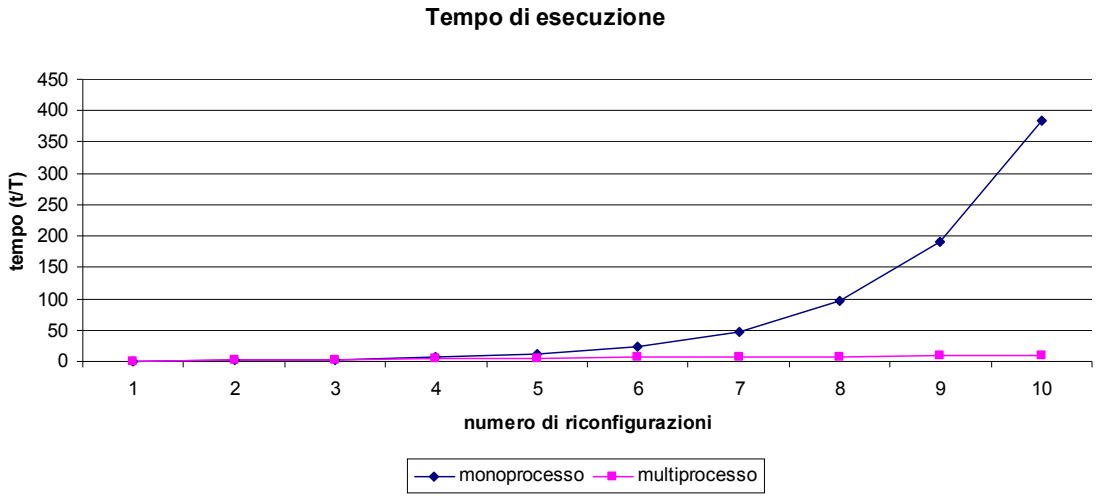


Figura 4.1. confronto tra il tempo di esecuzione necessario con il simulatore tuning monoprocesso e quello multiprocesso

Il simulatore è stato inoltre modificato per accettare in ingresso i parametri necessari a modificare la tipologia di metrica di esplorazione da utilizzare:

- -explore:metric
- -explore:pollref
- -wayadapt:tollerance

I due parametri del gruppo *explore* riportati permettono di selezionare la metrica di esplorazione con cui effettuare il tuning e la grandezza da utilizzare come trigger per l'esecuzione dell'algoritmo (hit, accessi, etc…).

Il paramentro tollerance del gruppo wayadapt permette invece di tarare la sensibilità della dell'algoritmo di esplorazione riguardo al range di tolleranza entro cui selezionare l'azione di riconfigurazione. Per pollrate pari a 100K hit, la tolleranza stabilità è dell'1%, mentre studi speramentali condotti nell'ambito del lavoro di tesi hanno dimostrato come per valori di K decrescenti, variare questo parametro linearmente con la riduzione del K, permette di mantenere una corretta sensibilità del metodo decisionale. Diminuendo il pollrate, i valori delle grandezze in gioco tendono ad essere sempre più piccoli, pertanto mantenere fissa la percentuale di tolleranza non permetteva di apprezzare le differenze della metrica tra i diversi rami, mentre scalarla eccessivamente non comportava aumenti significativi nell'efficienza dell'algoritmo.

Le modifiche relative al tuning multi processo sono state inserite anche nel simulatore per sistemi multiprogrammati.

4.3 Simulatore way-adapting differenziale normalizzato

L'esecuzione dell'algoritmo way-adapting è demandata ad una diversa implementazione del simulatore rispetto al tuning. In questo simulatore sono stati implementati anche gli algoritmi senza soglie (cfr. 6.4), accessibili tramite la configurazione di appositi parametri in fase di avvio delle simulazioni. In particolare, i parametri inseriti sono:

- -wayadapt:normalize
- -nothrlds:enable
- -nothrlds:type

Il parametro normalize, del gruppo wayadapt, permette di selezionare se utilizzare o meno la versione differenziale normalizzata dell'algoritmo way-adapting [7], mentre il gruppo di parametri nothrlds permette di utilizzare gli algoritmi senza senza soglie (enable) e di selezionare con quale algoritmo eseguire la simulazione (type).

4.4 Configurazione della cache

In questa sezione sono descritte in dettaglio le configurazioni utilizzate durante le simulazioni, relative alla cache L2 di tipo di D-NUCA e alla tecnica di riconfigurazione way-adapting. La stessa configurazione è stata utilizzata anche durante la sperimentazione degli algoritmi senza soglie. Si fa esplicito riferimento alle opzioni dei simulatori che sono stati introdotti precedentemente. Per questioni di spazio si tralasciano le descrizioni delle configurazioni del processore e del sottosistema di memoria (cache L1 e RAM).

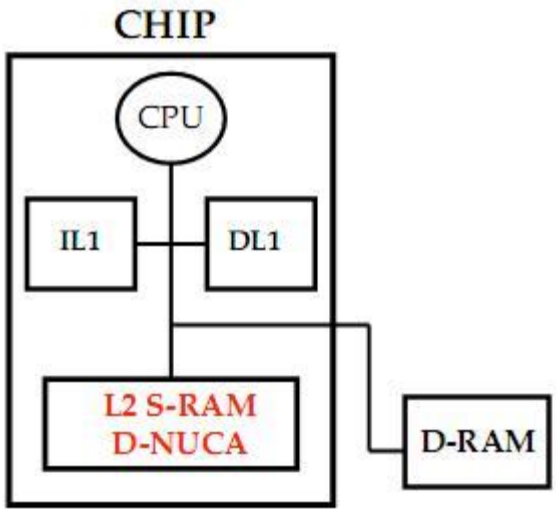


Figura 4.2. Architettura della memoria a livelli

Per valutare il comportamento della tecnica way-adapting al variare dei suoi parametri di funzionamento e successivamente la famiglia di algoritmi senza soglie, è stata utilizzata la stessa configurazione scelta per il lavoro di tesi [7]. La tecnologia selezionata per questo lavoro è quella con feature size di 70 nm, valore tipico di ancora molti dei processi costruttivi attuali. La cache D-NUCA utilizzata nelle simulazioni è di 8 MB, con 128 banchi, organizzati in una matrice di 16 righe (bank set) e 8 colonne, implementando quindi una cache 8 way set-associative. La scelta di questa particolare tipologia, è dovuta anche al fatto che l'algoritmo way-adapting risulta maggiormente efficace in presenza di un'alta associatività. A differenza del lavoro di tesi [7], in questo lavoro di tesi si è usata una cache con latenza pari a 300 cicli di clock: tutte le simulazioni effettuate, anche quelle di riferimento, sono state rieseguite con il nuovo parametro, in modo da essere tra loro confrontabili.

Per l'implementazione di questa cache, è stato scelta di implementare la politica simple mapping per il posizionamento dei blocchi, permettendo di identificare un indirizzo fisico che si presenta al controller della D-NUCA, come mostrato in figura [4.3]:

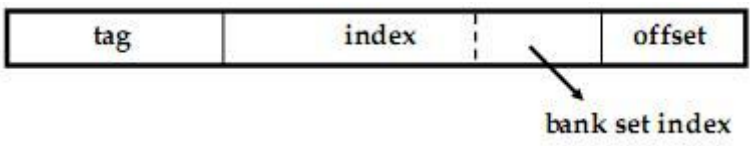


Figura 4.3. Indirizzo fisico di una cache D-NUCA

Facendo riferimento alla cache D-NUCA da 8 MB, con tecnologia a 70 nm, i singoli campi hanno il seguente significato: la parte di offset è necessaria per l'identificazione della word all'interno del blocco e consta di 6 bit (cioè $\log_2(64)$), essendo i blocchi di 64 byte; la parte meno significativa del campo index (bank set index) determina il bank set di appartenenza e, in questo caso, consta di 4 bit ($\log_2(16)$), poiché sono 16 i bank set; la parte più significativa del campo index determina invece l'indice del set all'interno del singolo banco e consta di 10 bit ($\log_2(1024)$, poiché ogni banco è costituito di 64 Kbyte e contiene esattamente 1024 set.

L'utilizzo dei bit meno significativi del campo index come bank set index, permette di effettuare un posizionamento interlacciato dei blocchi, dove due qualsiasi indirizzi consecutivi saranno sempre posizionati in blocchi diversi. In questo modo è possibile ridurre notevolmente i conflitti.

La politica di search utilizzata è la multi cast search.

Viene invece utilizzata come politica di promotion la generational promotion, con il trasferimento del blocco per il quale si verifica una hit nel banco immediatamente precedente tra quelli componenti il bank set.

È stato scelto di utilizzare la tail insertion come politica di inserimento dei nuovi blocchi.

Occorre precisare che la rete di collegamento tra gli switch e il controller della D-NUCA non prevede collegamenti verticali tra gli switch, se non per quelli corrispondenti alla prima colonna di banchi, come si può vedere in figura [4.4].

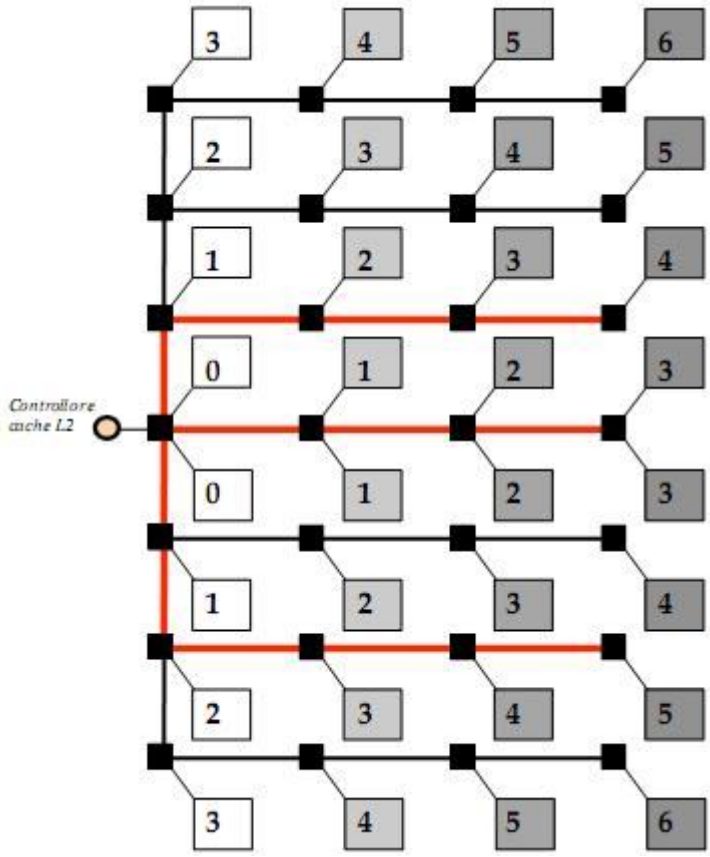


Figura 4.4. Routing di una cache D-NUCA

Ogni collegamento è bidirezionale, ovvero composto da due canali completamente separati da 128 bit per trasportare dati e indirizzi. I conflitti sono quindi limitati dalla totale separazione del traffico nelle due direzioni.

Il routing dei pacchetti prevede che, dato un indirizzo, il dato venga prima inviato sulla riga che potrebbe contenere il dato attraverso i collegamenti verticali e successivamente inoltrato ai banchi del relativo bank set, attraverso i link orizzontali (percorsi in rosso). Sempre in figura [4.4], è riportato il ritardo di introdotto dalle interconnessioni per ogni banco della cache D-NUCA nel caso di assenza di conflitti. In appendice A si trova il file di configurazione di sim-alpha utilizzato per definire l'architettura cache descritta fino a questo momento.

4.5 Benchmark

I benchmark utilizzati in questo lavoro di tesi per la valutazione dei comportamenti dell'algoritmo way-adapting e per lo studio degli algoritmi privi di

soglie, appartengono alle suite SPEC Cpu 2000 e NPB [11]..I benchmark del gruppo SPEC sono divisi in un due categorie:

- CINT2000: per applicazioni con aritmetica intera
- CFP2000: per applicazioni con aritmetica floating-point

I benchmark utilizzati nelle simulazioni eseguite in questo lavoro sono riportati nelle tabella seguente, dove, per ognuno di questi, sono indicati gli intervalli di simulazione assunti per ogni applicazione. Tali intervalli sono stati definiti da *Kim et al.* in [1], identificando nella fase di RUN, la sequenza di istruzioni che determina la maggiore attività di ogni benchmark. Tali fasi sono state identificate plottando il missrate della cache L2 dell'intera simulazione di ogni benchmark.

CINT2000	FFWD	RUN
Bzip2	744M	1.0B
Mcf	2.367B	300M
Parser	3.709B	200M
Perlbmk	5.0B	200M
Twolf	511M	200M

CFP2000	FFWD	RUN
Applu	267M	650M
Art	2.2B	200M
Equake	4.459B	200M
Galgel	4.0B	200M
Mesa	570M	200M
Mgrid	550M	1.06B

NPB	FFWD	RUN
Bt	800M	650M
Cg	600M	200M
Sp	2.5B	200M

I campi FFWD e RUN indicano rispettivamente il numero di istruzioni utilizzate per il fast forwarding e quello per la fase di esecuzione.

Durante le simulazioni, per ogni benchmark, è stata utilizzata la rispettiva eio trace (external I/O trace). Questo tipo di tracce, generate attraverso l'utilizzo del simulatore sim-eio, catturano lo stato iniziale dei programmi e tutte le seguenti interazioni che il programma in esame ha con il sistema operativo. In questo modo, una stessa simulazione eseguita diverse volte, ma con la medesima traccia eio, restituirà sempre lo stesso comportamento, svincolando così i test effettuati dalle aleatorietà che deriverebbero dal far gestire al sistema operativo ospite le system call effettuate dal benchmark simulato. Sim-alpha, come anche simplescalar da cui è derivato, non fa infatti full system simulation, delegando al

sistema operativo della macchina host la risoluzione delle system call: in questo modo l'utilizzo delle tracce eio, permette di utilizzare sim-alpha producendo simulazioni che non risentono del carico della macchina host.

5. Risultati

5.1 Analisi del comportamento delle soglie esistenti al variare dei parametri di funzionamento dell'algoritmo way-adapting

La prima analisi condotta durante il lavoro di tesi è stata relativa al comportamento delle soglie esistenti al variare dei parametri di funzionamento dell'algoritmo way-adapting. Lo scopo di questo studio è stato verificare l'esistenza o meno di un punto di funzionamento che ottenesse prestazioni migliori di quelle ottenute con pollrate pari a 100K hit.

Le simulazioni sono state condotte sia per le soglie di Kobayashi [1], che per le soglie ottenute attraverso l'algoritmo way-adapting differenziale normalizzato, introdotto con il lavoro di tesi [7].

I valori di pollrate interessati in questa fase sono stati 100K, 75K, 50K, 10K e 1000 hit .

Per ogni coppia di soglie, vengono riportati i valori di IPC e associatività media per ogni benchmark utilizzato e la media di tali valori, separatamente per i benchmark della suite SPEC e della suite NAS.

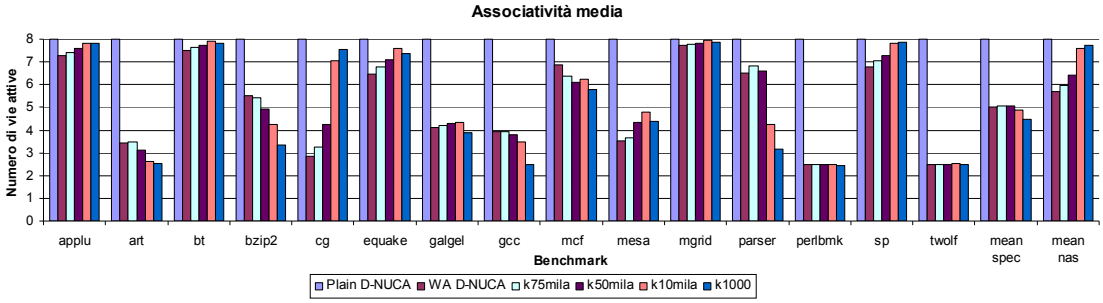


Figura 5.1 a. associatività media per la coppia di soglie [0.005; 0.02]

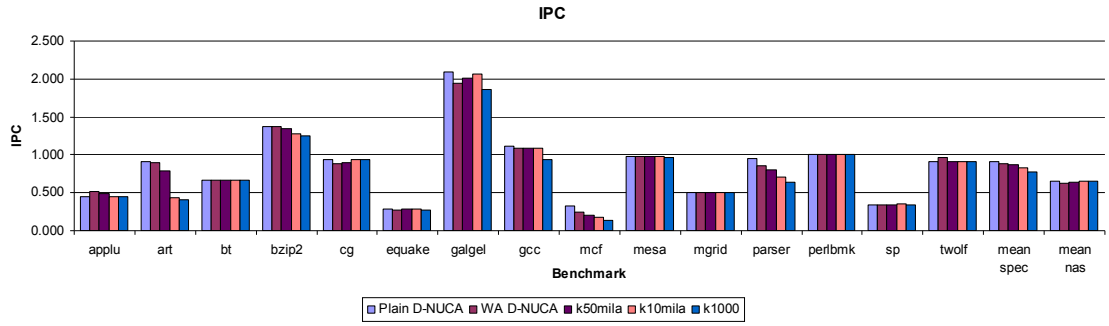


Figura 5.1 b. IPC per la coppia di soglie [0.005; 0.02]

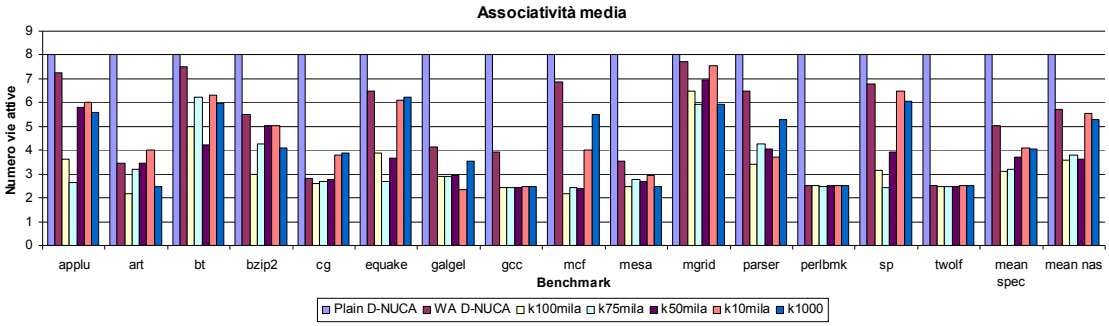


Figura 5.2 a. associatività media per la coppia di soglie [-1.2; 0.6]

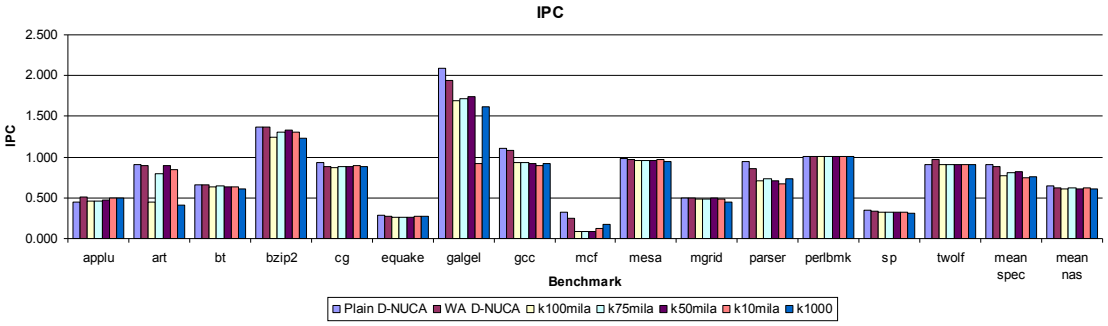


Figura 5.2 b. IPC per la coppia di soglie [-1.2; 0.6]

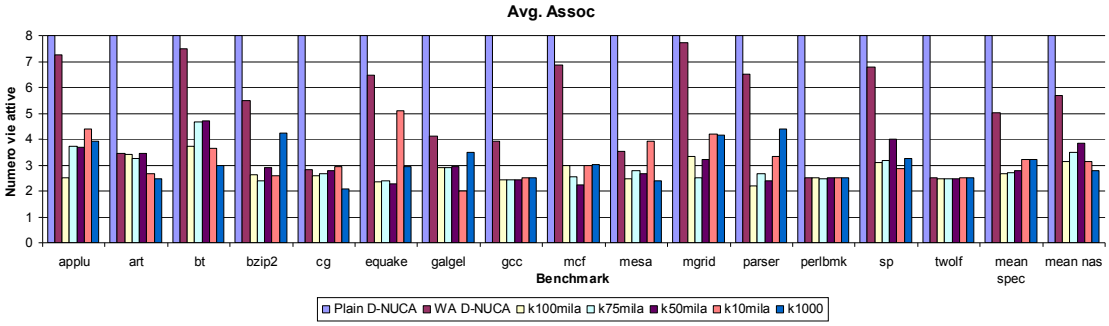


Figura 5.3 a. associatività media per la coppia di soglie [-0.6; 0.33]

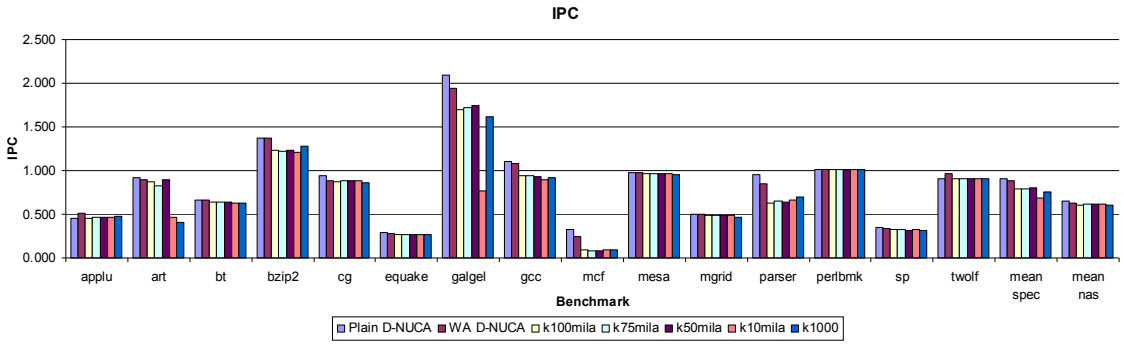


Figura 5.3 b. IPC per la coppia di soglie [-0.6; 0.33]

Nei grafici qui proposti, le simulazioni con valori di pollrate variato (gruppi di destra nei grafici) sono confrontati i risultati ottenuti con cache di tipo Plain D-NUCA (ovvero cache in cui non è attivo l'algoritmo way-adapting) e cache in cui è attivo l'algoritmo way-adapting originale (WA D-NUCA). Queste due tipologie di simulazioni verranno usate come riferimento in tutti i grafici seguenti.

Il comportamento dei singoli benchmark al variare del pollrate presenta situazioni diverse, a seconda del gruppo a cui questi appartengono e dalla località dei loro dati. Alcuni benchmark infatti presentano un incremento dell'associatività media unito ad una diminuzione dell'IPC, mentre altri presentano tendenze divergenti dei due parametri. Tutto questo comporta un allontanamento delle prestazioni medie dal trade-off ottimale. E' possibile ad esempio notare come per la coppia di soglie [0.005; 0.02], l'IPC medio dei benchmark del gruppo NAS tenda a rimanere costante, ma la rispettiva associatività media tenda ad aumentare.

Tuttavia, uno dei fattori che contribuiscono a questo allontanamento dalle condizioni ottimali del trade-off prestazionale, è un comportamento comune a tutti i benchmark, dovuto alle condizioni di funzionamento dell'algoritmo way-adapting. Tale comportamento consiste in un fenomeno di oscillazione della cache, dovuto ad un aumento di sensibilità dell'algoritmo al diminuire del valore di pollrate, che può introdurre una sorta di rumore che altera le riconfigurazioni ottimali: questo fenomeno veniva invece filtrato con un campionamento più lasco.

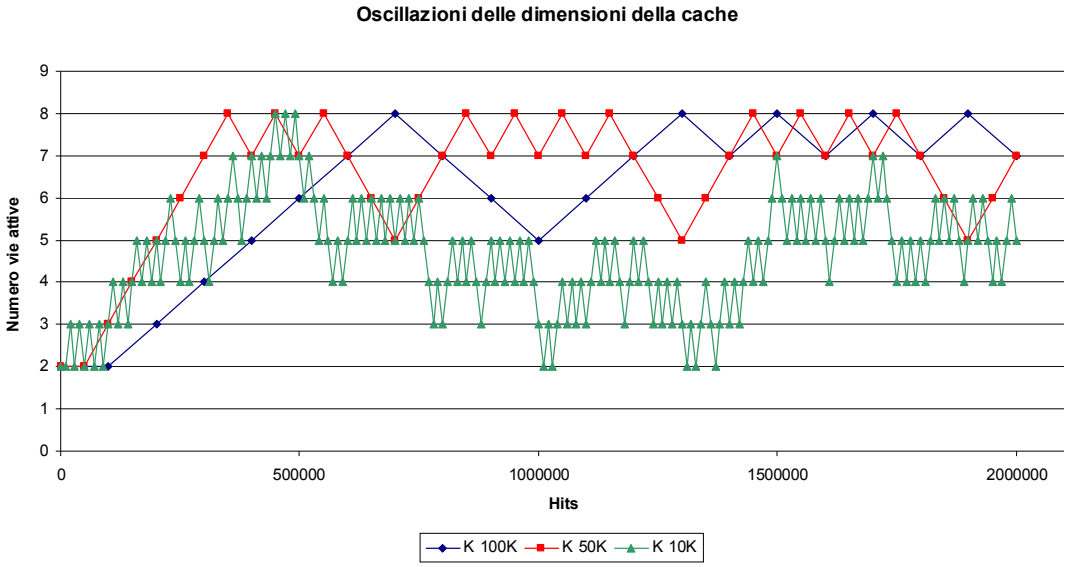


Figura 5.4. Oscillazione della cache per valori di K pari a 100K, 50K e 10K hit per il benchmark parser

In figura [5.4] è possibile vedere chiaramente il fenomeno di oscillazione e come questo mantenga la dimensione della cache a valori più bassi di quelli realmente richiesti dal benchmark in esecuzione.

All'attivazione di un nuova via, dovuta alla decisione di un'azione di riconfigurazione, questa risulta vuota, ovvero le linee sono tutte in stato non valido, in quanto la tecnica gatedV_{dd} non mantiene il dato. Se il valore di pollrate è troppo piccolo e il benchmark non riesce a trarre subito beneficio dalla via appena accesa, il contatore delle hit sull'ultima via attiva non riesce ad assumere valori significativi, decretando lo spegnimento delle via appena accesa.

5.2 Ricerca del valore di pollrate ottimale

Lo studio delle soglie esistenti con valori diversi di pollrate, ha suggerito l'opportunità di dover rieseguire il tuning, nel caso in cui i parametri di funzionamento vengano variati. In questa fase del lavoro, è stata posta l'attenzione sulla ricerca di valore ottimale del parametro K.

Per determinare quale fosse il valore di pollrate ottimale è stato inizialmente studiato il tuning per differenti valori di K. Il tuning infatti permette di identificare un cammino ottimale lungo l'albero delle riconfigurazioni, inseguendo l'ottimalità di una data metrica. Di seguito sono mostrati i grafici di IPC e associatività media per valori di pollrate pari a 100K, 50K e 10K hit, confrontati con i risultati ottenuti per le soglie conosciute con l'utilizzo dell'algoritmo way-adapting.

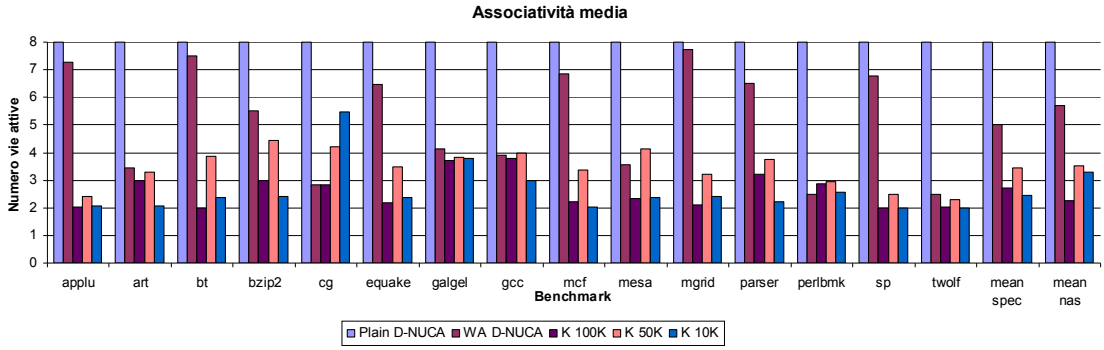


Figura 5.6 a. Associatività media per simulazioni tuning con metrica di esplorazione gmr e differenti valori di K (100K, 50K e 10K hit)

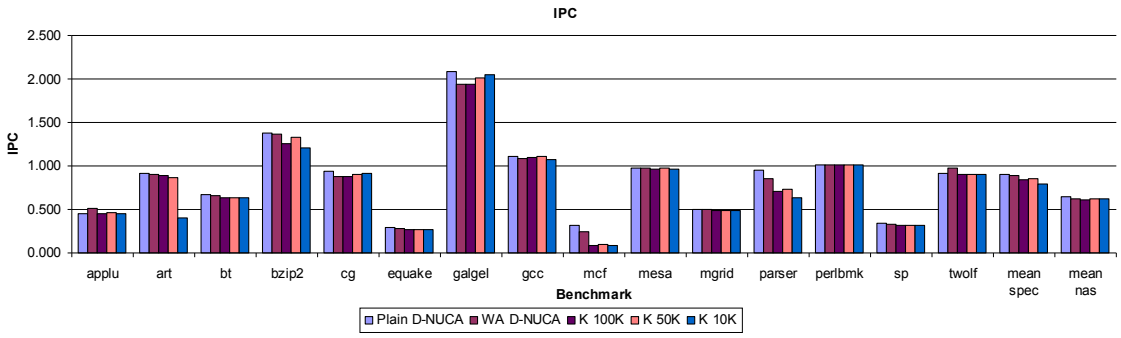


Figura 5.6 b. IPC per simulazioni tuning con metrica di esplorazione gmr e differenti valori di K (100K, 50K e 10K hit)

Nei grafici è possibile vedere come l'utilizzo di un valori di pollrate di 100K hit, offra un trade-off migliore rispetto a valori minori.

Viene così dimostrato come K = 100K hit è il compromesso ottimale in questa situazione. Tuttavia ciò non basta per dimostrare che tale valore sia ottimale in assoluto: per ottenere questo risultato è necessario prima dimostrare che non esista un valore di K diverso da 100K, che abbia comportamenti migliori con una metrica di esplorazione diversa.

Per poter confrontare metriche diverse, è necessario che tutti gli altri parametri di funzionamento siano fissi, pertanto è stato deciso di utilizzare K = 100K hit, per poter avere anche un termine di confronto con le simulazioni Plain D-NUCA e WA D-NUCA. In figura [5.7 a] e [5.7 b] sono mostrati i grafici di IPC e associatività media per le simulazioni tuning effettuate con metriche grm, lmr, gipc e lipc.

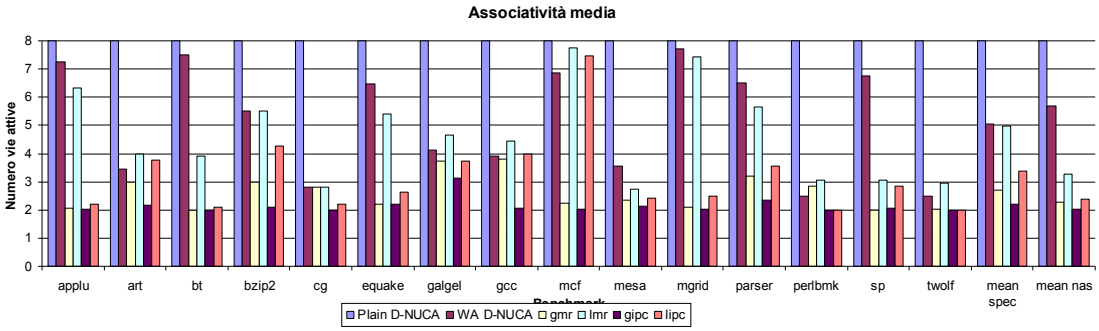


Figura 5.7 a. Associatività media per simulazioni tuning con K = 100K hit e metriche di esplorazione gmr, lmr, gipc, lipc

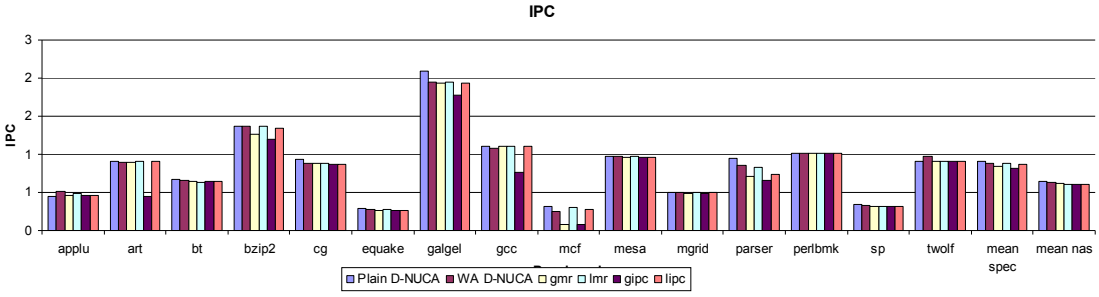


FIgura 5.7 b. IPC per simulazioni tuning con K = 100K hit e metriche di esplorazione gmr, lmr, gipc, lipc

Dai grafici si evince come le metriche che utilizzano valori dei parametri locali alla singola riconfigurazione ottengano prestazioni migliori. Tra le due metriche di tipo locale, quella che utilizza l'IPC (lipc) è risultata ottenere il miglior comportamento.

Identificata in questo modo la metrica più performante per le simulazioni tuning, è possibile adesso verificare se K = 100K hit sia o meno il valore ottimale di pollrate.

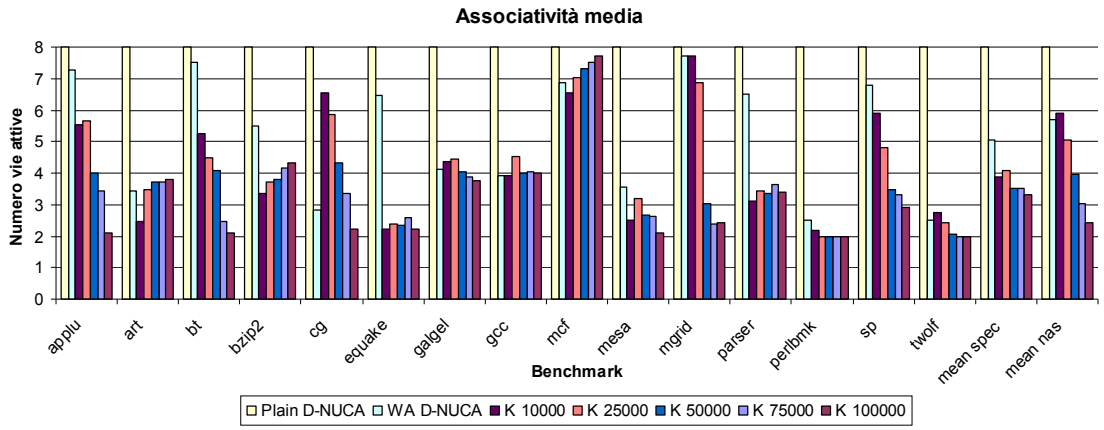


Figura 5.8 a. Associatività media per simulazioni tuning con metriche di esplorazione lipc e differenti valori di K

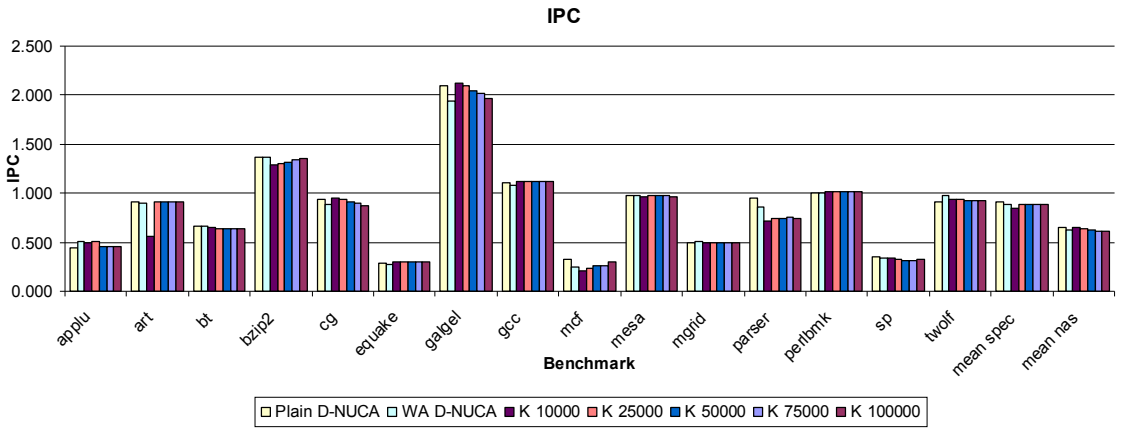


Figura 5.8 b. IPC per simulazioni tuning con metriche di esplorazione lipc e differenti valori di K

In figura [5.8 a] e [5.8 b] sono mostrati i risultati ottenuti applicando valori differenti di K alla metrica di esplorazione ipc locale. E' chiaro come anche in questo caso K = 100K hit sia risultato essere il valore di pollrate che ha ottenuto le migliori performance.

A questo punto è possibile asserire che K = 100K hit sia il valore di pollrate ottimale da applicare all'algoritmo way-adapting.

5.3 Estrazione delle soglie con metodi di calcolo alternativi

Durante lo studio dell'ottimalità del pollrate sono state estratte alcune coppie di soglie con lo scopo di verificare il comportamento dell'algoritmo way-adapting differenziale con valori calcolati ad-hoc per un determinato valore di pollrate. Per l'estrazione di queste coppie di soglie sono stati utilizzati i metodi di estrazione alternativi illustrati precedentemente (cfr. metodo del conteggio e metodo del clustering). Questi metodi alternativi permettono di estrarre velocemente coppie di soglie anche in presenza di un numero elevato di vincoli, come può accadere utilizzando valori di pollrate molto bassi.
Di seguito verranno mostrati due esempi, uno per ogni metodo alternativo presentato, che mostrano i risultati delle simulazioni ottenute utilizzando i valori estratti con queste tecniche.

5.3.1 Utilizzo del metodo del conteggio

Come descritto precedentemente (cfr. 4.2), il metodo del conteggio identifica le soglie in base alla massimo numero di vincoli di riconfigurazione attivi (accensioni e spegnimenti) soddisfatti.

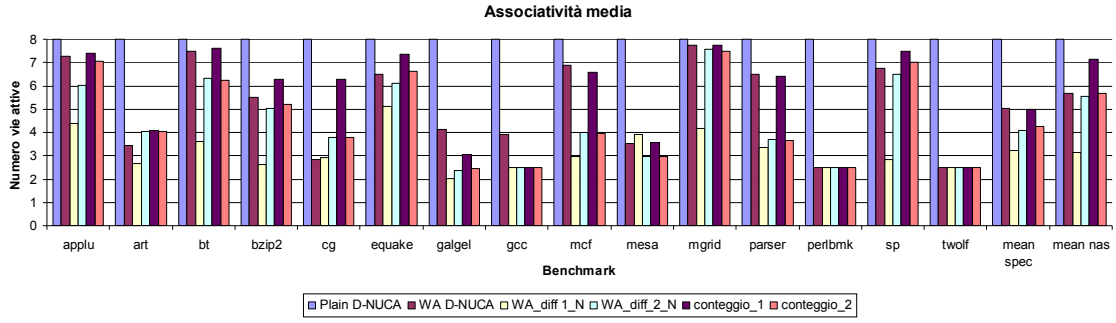


Figura 5.9 a. Associatività media per le coppie di soglie estratte con il metodo del conteggio per K = 10K hit (conteggio_1 [-0.608; -0.598], conteggio_2 [-0.608; 0.1995])

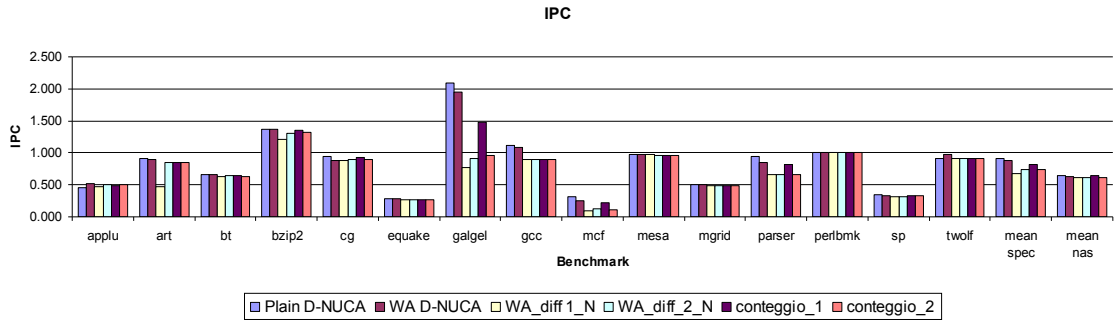


Figura 5.9 b. IPC per le coppie di soglie estratte con il metodo del conteggio per K = 10K hit (conteggio_1 [-0.608; -0.598], conteggio_2 [-0.608; 0.1995])

Nei grafici sono mostrati i valori di associatività media e IPC ottenuti con le soglie estratte, per K = 10K hit, con il metodo del conteggio. Il confronto viene effettuato con i valori delle simulazioni per le soglie euristiche e per quelle calcolate con il metodo del way-adapting differenziale normalizzato, ottenuti per K = 100K hit, avendo visto che per tali valori il comportamento dell'algoritmo è ottimale.

E' possibile notare come le soglie estratte abbiamo un buon comportamento in termini di IPC, ma l'algoritmo richiede un numero di via attive pari (e a volte superiore) a quello ottenuto con il way-adapting tradizionale. Per la coppia di soglie conteggio_1, è evidente come questo problema sia legato al poco margine tra le due soglie, che non permette di ottenere facilmente delle fasi di mantenimento, facendo oscillare continuamente la cache. Rilasciando alcuni vincoli, ovvero considerando valori della metrica D nell'intorno di quelli identificati con il metodo del conteggio, si può ottenere un comportamento migliore in termini di associatività media, tuttavia rinunciando a parte delle prestazioni in IPC (conteggio_2 nel grafico).

5.3.2 Utilizzo del metodo del clustering

Nel lavoro di tesi, il metodo del clustering è stato utilizzato principalmente per avere un riscontro immediato della distribuzione delle riconfigurazioni nello spazio.

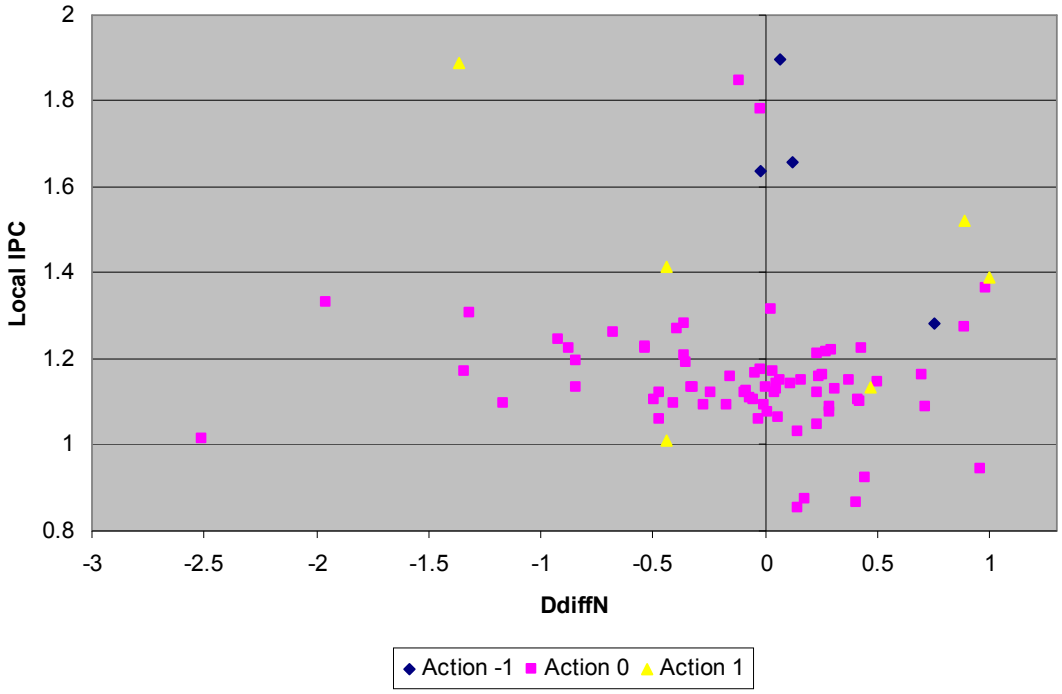


Figura 5.10. Grafico della distribuzione delle riconfigurazioni

In figura [5.10] è possibile vedere chiaramente come le hit si distribuiscano nello spazio senza che sia possibile identificare una precisa separazione tra i diversi tipi di riconfigurazione. Analizzando questo grafico è semplice capire che utilizzando il metodo delle disquazioni saremo costretti a rilasciare un certo numero di vincoli.

Se con il metodo del conteggio esiste la possibilità di ottenere un'area ristretta in cui l'algoritmo sceglierà un'azione di mantenimento, con questo metodo è possibile tarare le soglie in base a quanto si vuole che l'algoritmo sia conservativo. E' stato detto infatti come con questo metodo sia possibile identificare le aree con la maggior concentrazione di vincoli (cfr. 4.3): con questo metodo è possibile posizionare graficamente le soglie, in modo che contengano l'area con maggior concentrazione di vincoli di mantenimento.

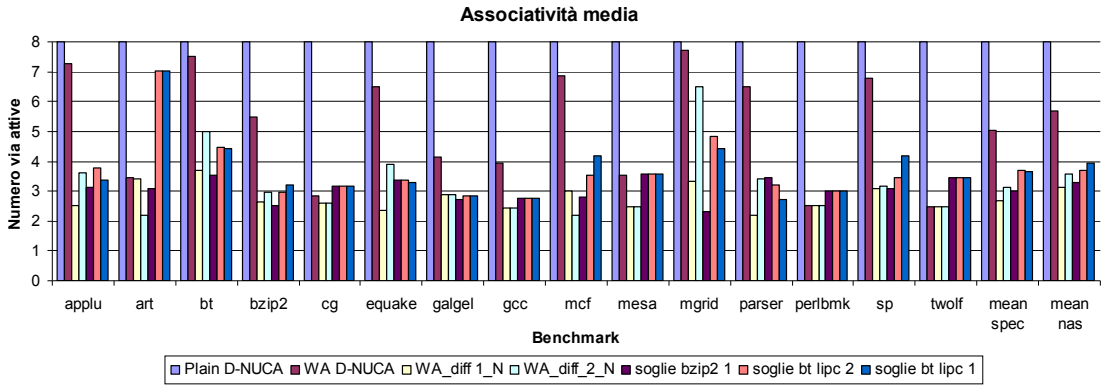


Figura 5.11 a. Associatività media per le coppie di soglie estratte con il metodo del clustering (soglie bzip2_1 [-0.5; 0.5], soglie bt lipc_1 [-0.04; 0.1279] soglie bt lipc_2 [-0.2; 0.2])

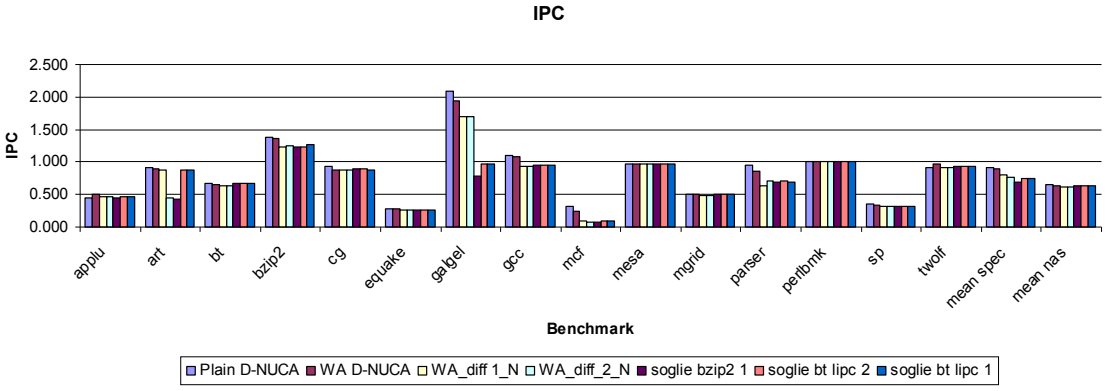


Figura 5.11 b. IPC per le coppie di soglie estratte con il metodo del clustering (soglie bzip2_1 [-0.5; 0.5], soglie bt lipc_1 [-0.04; 0.1279] soglie bt lipc_2 [-0.2; 0.2])

I risultati ottenuti con queste soglie non sono importanti per i loro valori globali, ma invece mostrano chiaramente come alcuni benchmark siano estremamente importanti per ottenere buone performance medie. Se infatti le prestazioni di alcuni benchmark, come ad esempio quelli dela suite NAS, risultano poco sensibili alla variazione delle soglie applicate, tra i benchmark SPEC è molto importante che le soglie siano tarate correttamente.

L'analisi di questi risultati ha portato all'identificazione di un set di benchmark, definiti critici, sulla cui ottimizzazione si è concentrata la seconda parte del lavoro di tesi, portando a progettare una famiglia di algoritmi senza soglie.

I benchmark identificati critici per le prestazioni globali delle simulazioni sono: art, bzip2, equake, galgel, gcc, mcf e parser.

5.4 Algoritmi senza soglie

Durante lo studio dell'ottimalità del parametro K, è stato verificato come alcuni benchmark abbiano un peso visibilmente maggiore nelle performance globali. Mentre alcuni benchmark, in particolare quelli del gruppo NAS, sono tendenzialmente insensibili alle soglie utilizzate, altri, come gcc, mcf, parser e altri, risentono sensibilmente di una configurazione non ottimale dei parametri dell'algoritmo.

Nel corso della ricerca del pollrate ottimale, l'analisi delle riconfigurazioni delle simulazioni tuning e l'ispezione della variazione delle grandezze a loro associate (metrica D, missrate, IPC) ha portato all'identificazione di una serie di fenomeni ricorrenti in questi benchmark (definiti critici), che legano le azioni di riconfigurazione a particolari variazioni delle grandezze.

Tra questi comportamenti è stato riscontrato come le azioni di riconfigurazione avvengano principalmente in corrispondenza di cambiamenti di segno della metrica D differenziale.

Analogamente, è stato visto come le azioni intraprese dalle simulazioni tuning siano strettamente legati ai cambiamenti di segno delle grandezze differenziali. Lo studio di questi comportamenti è stato quindi esteso a tutti i benchmark utilizzati, per verificare che fosse comune a tutto il working set.

Dall'analisi di questi comportamenti è stata progettata una famiglia di algoritmi che non prevede l'utilizzo delle soglie, realizzando metodi decisionali che tengono conto solamente delle transizioni di segno delle grandezze differenziali.

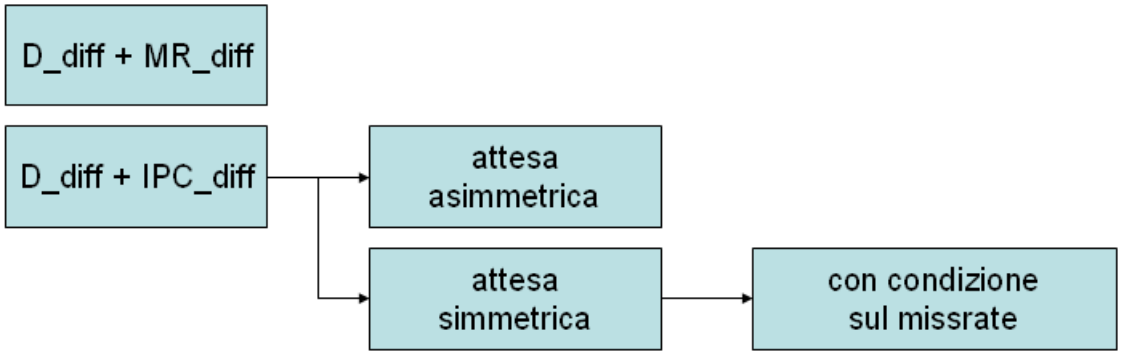


Figura 5.12. Algoritmi senza soglie

In figura [5.12] è mostrato il percorso seguito nella progettazione degli algoritmi senza soglie. Di seguito sono analizzati singolarmente, riportando in pseudo codice il loro principio di funzionamento.

5.4.1 Algoritmi di origine

I primi due algoritmi progettati utilizzano le metrica D differenziale e, rispettivamente, le grandezze IPC differenziale e missrate differenziale. Tutte le grandezze utilizzate sono calcolate localmente al punto di riconfigurazione, visto il miglior comportamento di queste grandezze dimostrato durante la ricerca del pollrate ottimale.

```
If (D_diff > 0 && MR_diff > 0) accendi una via
If (D_diff < 0 && MR_diff < 0) spegni una via
Else mantieni
```

Tabella 1. metrica D differenziale (D_diff) e missarate differenziale (MR_diff)

```
If (D_diff > 0 && IPC_diff < 0) accendi una via
If (D_diff < 0 && IPC_diff > 0) spegni una via
Else mantieni
```

Tabella 2. metrica D differenziale (D_diff) e IPC differenziale (IPC_diff)

Nelle tabelle è mostrato lo pseudo codice dei primi due algoritmi derivati dallo studio delle riconfigurazioni del tuning.

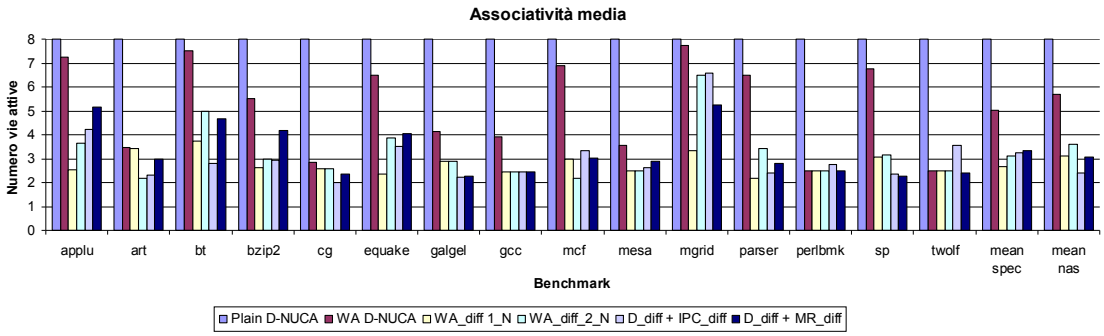


Figura 5.13 a. Associatività media per gli algoritmi senza soglie originali

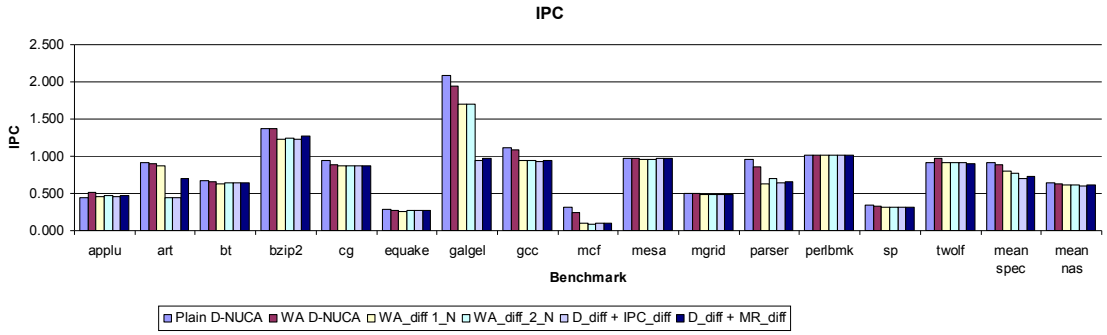


Figura 5.13 b. IPC media per gli algoritmi senza soglie originali

L'utilizzo di questi algoritmi permette di migliorare il comportamento di alcuni dei benchmark che incidono maggiormente sulle prestazioni globali. Tuttavia anche questa coppia di algoritmi è affetta dal fenomeno di oscillazione della cache: in questo caso non è il valore di pollrate che causa il fenomeno, bensì l'eccessiva sensibilità dell'algoritmo, dovuta all'utilizzo dei soli cambi di segno delle grandezze coinvolte nella decisione.

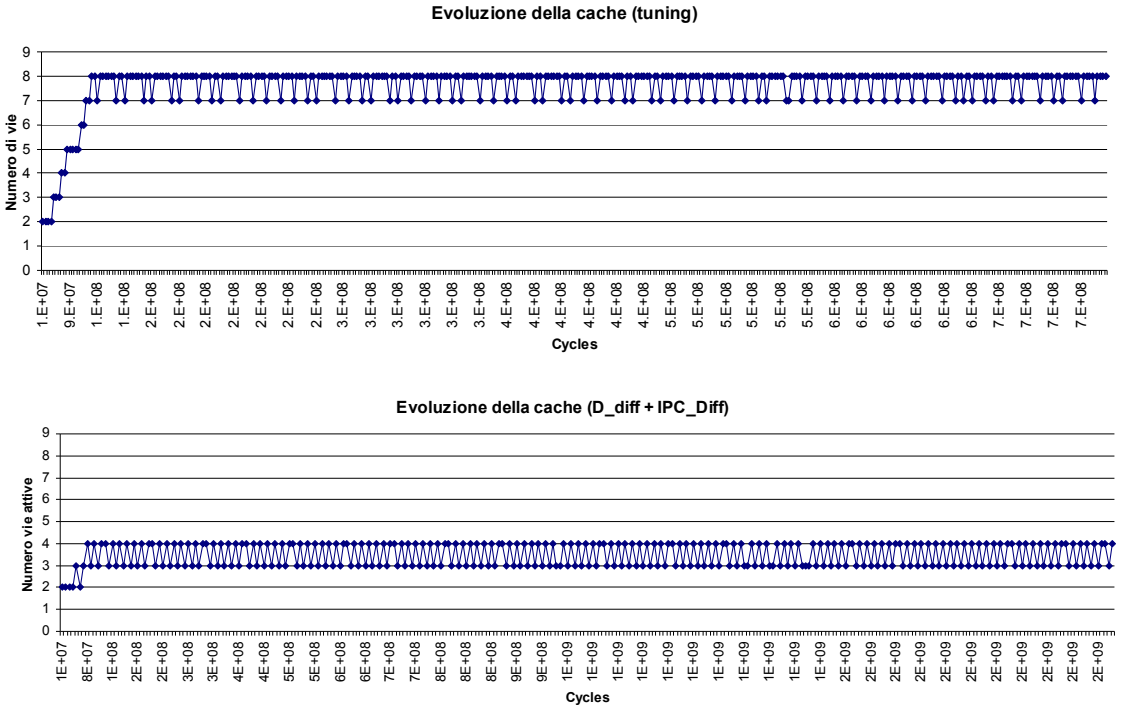


Figura 5.14. Confronto dell'evoluzione della cache tra il tuning e l'algoritmo senza soglie D_diff + IPC_diff per il benchmark mcf

Le figura [5.14] mostra l'evoluzione della cache del benchmark mcf (per cui è stato notato come per la fase presa in considerazione sia necessaria una cache con ampia associatività per ottenere prestazioni migliori) per la simulazione tuning e per una simulazione effettuata con il benchmark in tabella 2. E' evidente come l'algoritmo non riesca ad inseguire il comportamento teorico, a causa dell'elevato numero di oscillazioni generato.
Per ovviare a tale fenomeno, è stato pensato di introdurre una condizione di attesa nell'algoritmo.

5.4.2 Algoritmi con condizione di attesa

Gli algoritmi con condizione di attesa affrontano il problema della sensibilità dell'algoritmo inserendo una condizione sull'azione di riconfigurazione. In questo ramo sono stati sviluppati due approcci diversi: attesa asimmetrica, dove la condizione è solamente sullo spegnimento e attesa simmetrica, dove la condizione è su entrambe le decisioni.

```
If (D_diff > 0 && IPC_diff < 0) accendi una via se l'azione
precedente era diversa da uno spegnimento
If (D_diff < 0 && IPC_diff > 0) spegni una via se l'azione
precedente era diversa da un'accensione
Else mantieni
```

Tabella 3. algoritmo con attesa simmetrica

```
If (D_diff > 0 && IPC_diff < 0) accendi una via
If (D_diff < 0 && IPC_diff > 0) spegni una via se l'azione
precedente era diversa da un'accensione
Else mantieni
```

Tabella 4. algoritmo con attesa asimmetrca

Di seguito sono riportate le evoluzioni della cache per il benchmark di riferimento mcf, valutate per i due algoritmi e confrontate con l'evoluzione ottenuta nella simulazione tuning.

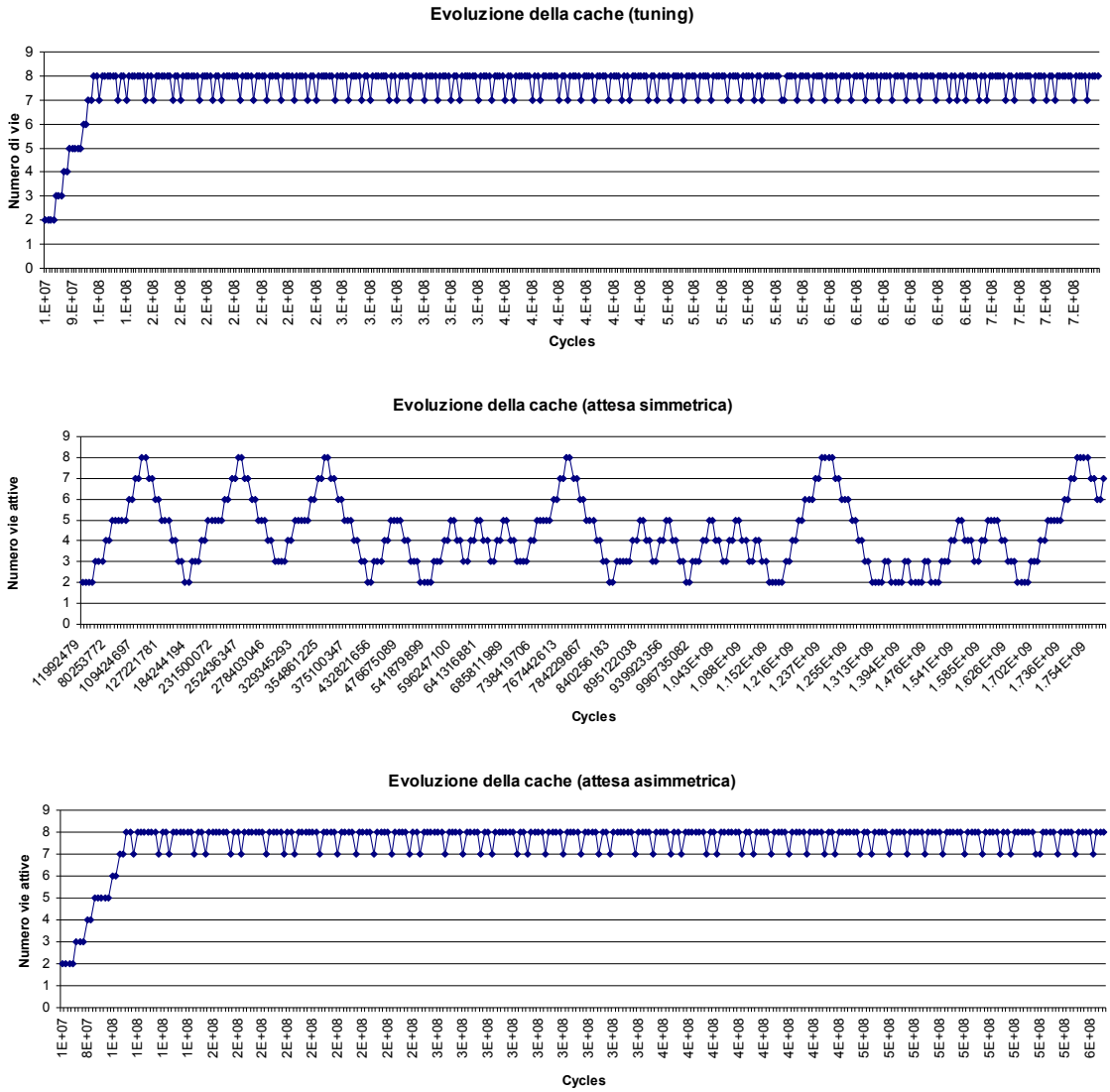


Figura 5.15. Confronto dell'evoluzione della cache tra il tuning e l'algoritmo senza soglie D_diff + IPC_diff con attesa simmetrica per il benchmark mcf

Come è possibile notare, l'algoritmo con attesa asimmetrica permette di riprodurre maggiormente il comportamento del tuning, riuscendo a inseguire le necessità del benchmark in termini di associatività.

L'algoritmo con attesa asimmetrica soffre però di un problema diverso: sebbene infatti riesca a ottenere un comportamento migliore, lo sbilanciamento verso l'accensione delle vie della cache, lo rende poco performante in termini di associatività media e, quindi, di consumo energetico.

5.4.3 Algoritmo con attesa simmetrica condizionata

Per ovviare al problema dello sbilanciamento dei benchmark verso una particolare decisione, è stato deciso di perfezionare il comportamento dell'algoritmo con attesa simmetrica, in quanto offre le stesse possibilità di accensione e spegnimento.
Analizzando ancora i dati prodotti dalle simulazioni tuning, è stato osservato che spesso ad una accensione non corrisponda una diminuzione del missrate: quando ciò avviene, è molto probabile che effettuare comunque un'accensione non sia una scelta corretta. Pertanto, nel decidere se effettuare o meno un'accensione, è stato inserita una condizione sul missrate, che comunque non necessità dell'utilizzo di soglie.

```
If (D_diff > 0 && IPC_diff < 0)
```

- accendi se l'azione precedente era diversa da uno spegnimento
- se l'azione precedente era un'accensione, accendi solamente se il missrate è diminuito rispetto al passo precedente

```
If (D_diff < 0 && IPC_diff > 0) spegni se l'azione
precedente era diversa da un'accensione
Else mantieni
```

Tabella 5. algoritmo con attesa simmetrica condizionata

L'utilizzo di questo algoritmo permette di ottenere le stesse prestazioni in termini di IPC dell'algoritmo con attesa asimmetrica, ma di ridurre l'associatività media.

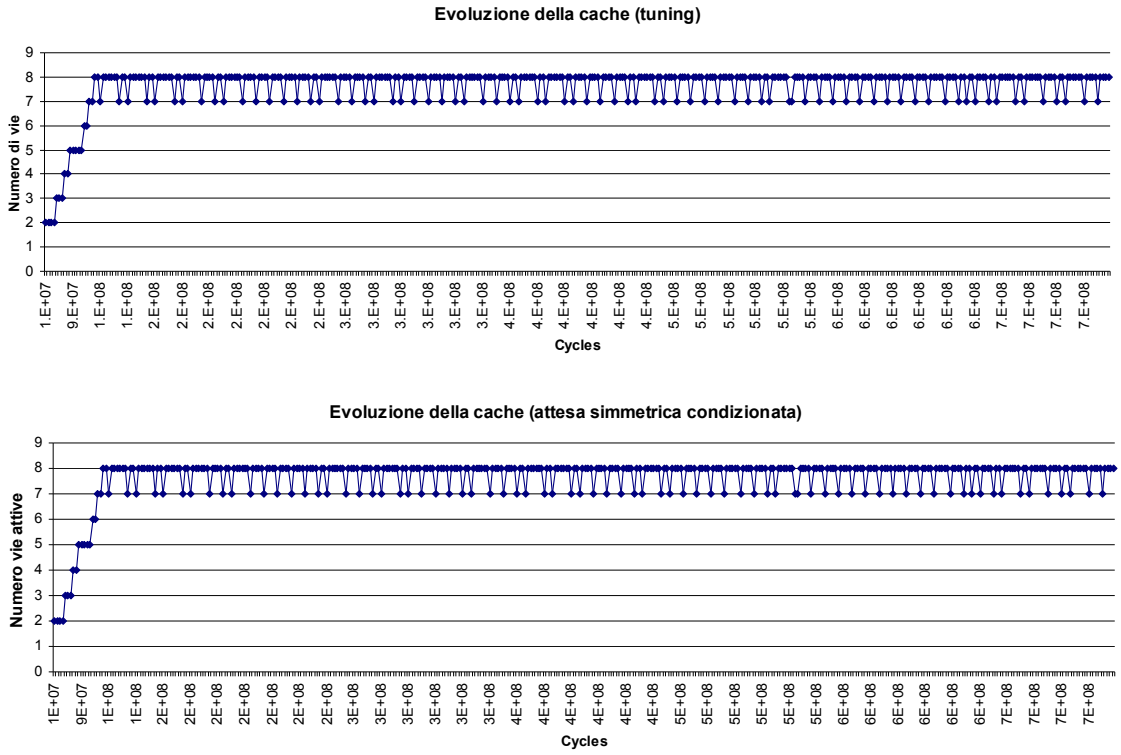


Figura 5.16. Confronto dell'evoluzione della cache tra il tuning e l'algoritmo senza soglie D_diff + IPC_diff con attesa simmetrica condizionata per il benchmark mcf

I grafici mostrano come questi algoritmi offrano buon prestazioni in termini di IPC, pur spegnendo maggiormente delle WA D-NUCA. Questa famiglia di algoritmi, ed in particolare quello con attesa simmetrica condizionata, si pone a livevvo di prestazioni in una posizione intermedia tra le soglie identificate da Kobayashi [3] e le soglie utilizzate nell'algoritmo way-adapting differenziale normalizzato [7], ma non prevede l'utilizzo di soglie e pertanto non necessita di un tuning offline.

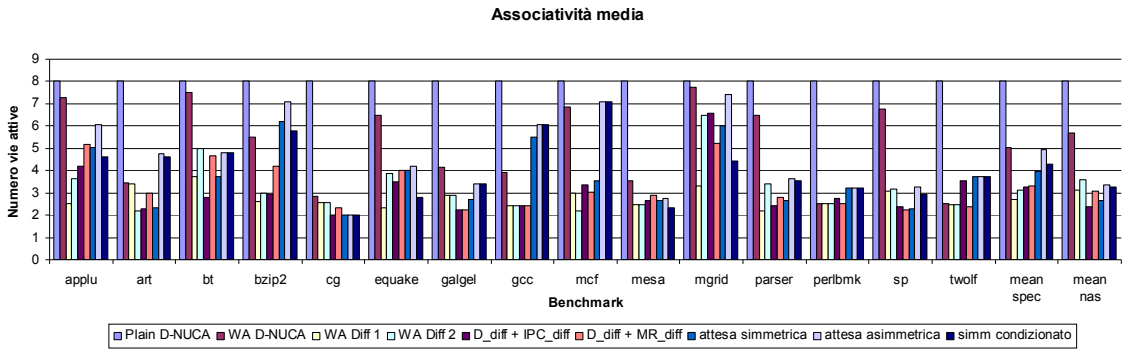


Figura 5.17 a. Associatività media per gli algoritmi senza soglie

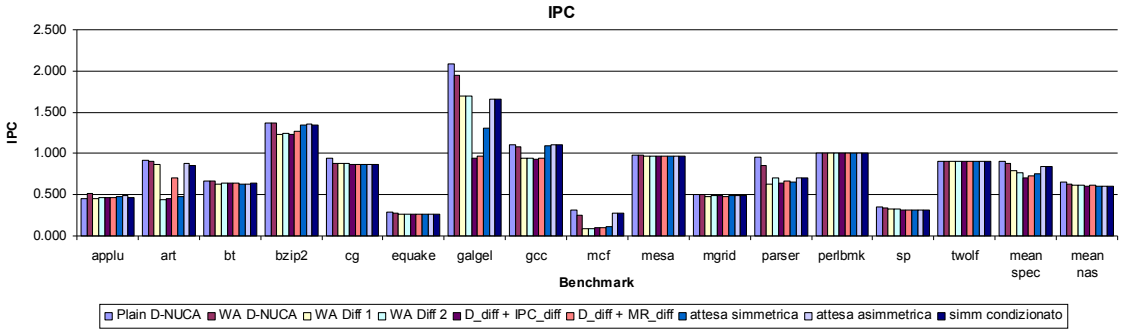


Figura 5.17 b. IPC per gli algoritmi senza soglie

Essendo però queste tecnologie rivolte a sistemi con bassi consumi di energia, è necessario confrontare sotto l'aspetto energetico questa nuova classe di algoritmi, con quelli precedenti.

I grafici seguenti mostrano, benchmark per benchmark, i valori del parametro EDP (Energy Delay Product) per gli algoritmi senza soglie e per i valori di riferimento Plain D-NUCA, WA D-NUCA. In questo caso l'EDP mostrato è stato stimato come il prodotto tra l'associatività media di un dato benchmark e il numero di cicli della relativa simulazione, il tutto normalizzato benchmark per benchmark, con l'EDP calcolato per le simulazioni con cache di tipo Plain D-NUCA. L'Energy Delay Product (EDP), è una metrica composta che è comunemente usata per valutare l'efficienza energetica: un'architettura per cui si ottenga un basso EDP è maggiormente efficiente perché in grado di consumare meno energia. Per la stima del consumo energetico, è stato utilizzato il numero di cicli della simulazione come riferimento temporale, mentre come riferimento energetico è stata utilizzata l'associatività media. L'utilizzo dell'associatività media è una buona approssimazione perché in questo tipo di cache l'energia statica è dominante, pertanto la componente dinamica è trascurabile rispetto alle correnti di leakage che attraversano i transistor che compongono la memoria.

$$EDP = (AvgAssoc_{Sim} * Cycles_{Sim}) / (AvgAssoc_{Plain} * Cycles_{Plain})$$

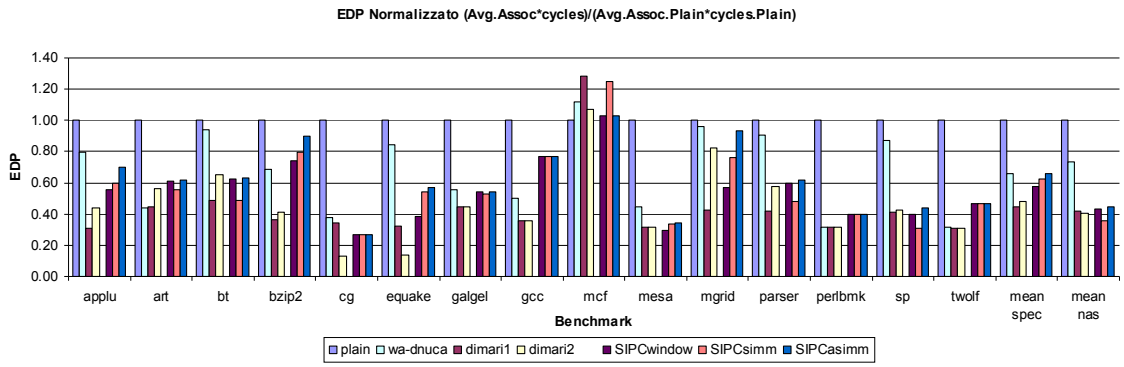


Figura 5.18. Confronto benchmark per benchmark del consumo energetico tra gli algoritmi senza soglie e gli algoritmi di riferimento

Gli algoritmi senza soglie offrono, mantenendo limitato il degrado prestazionale, una buona riduzione del consumo energetico rispetto alle cache di tipo Plain D-NUCA e al way adapting tradizionale.

Come è possibile vedere dal grafico [5.18], i benchmark che incidono maggiormente sul consumo energetico, sono esattamente quelli che sono critici nel degrado delle prestazioni. Tentare di migliore il comportamento prestazionale di tali benchmark, porta necessariamente ad un aumento del consumo energetico. Un caso evidente della necessità di questo trade-off è proprio mcf dove l'elevato consumo energetico è causa dell'espansione della cache, però necessaria per ottenere delle buone performance dal benchmark.

Questi algoritmi dimostrano però come sia possibile ottenere comunque una riduzione del consumo di potenza, mantenendo un degrado minimo delle prestazioni, senza l'utilizzo di soglie e senza quindi avere la necessità di effettuare una progettazione offline delle stesse.

6. Conclusioni

Nel corso del lavoro di testi sono stati determinati i parametri di funzionamento ottimali per l'algoritmo way-adapting. Per raggiungere questo obiettivo è stata realizzata una nuova metodologia di tuning, in grado di elaborare parallelamente le scelte possibili ad ogni ramo dell'albero delle riconfigurazioni, riducendo drasticamente in questo modo il tempo di calcolo necessario ad effettuare una singola simulazione e permettendo così di effettuare esplorazioni con valori di K molto bassi. La ricerca del K ottimale ha inoltre evidenziato come utilizzare delle tecniche di esplorazione basate su valori locali delle grandezze associate alle riconfigurazioni (IPC locale, missrate locale), porti alla costruzione di un percorso più performante.

L'analisi delle grandezze associate alle riconfigurazioni delle simulazioni tuning (metrica D, missrate, IPC) ha permesso di identificare una serie di comportamenti ricorrenti in tutti i benchmark, che hanno portato alla progettazione di una famiglia di algoritmi che non richiede l'utilizzo delle soglie.

Gli algoritmi sviluppati in questa seconda parte del lavoro non risentono della degradazione dell'IPC riscontrata con la tecnica del way-adapting differenziale, pur mantenendo ridotto il consumo energetico della cache e svincolandosi dalla necessità di dover calcolare e specificare una coppia di soglie.

A partire dai risultati degli algoritmi senza soglie presentati in questo lavoro, un possibile sviluppo futuro è cercare di realizzare un algoritmo della stessa famiglia in grado di ottimizzare anche il consumo di potenza. E' dimostrato come il tuning sia anche ottimizzato rispetto al consumo di potenza, quindi può essere una soluzione valida quella di continuare l'analisi delle variazioni delle grandezze associate alle riconfigurazioni tuning, per identificare ulteriori comportamenti comportamenti ricorrenti.

Appendice A. Configurazione della cache D-NUCA

```
### 08MB_16x8_70nm.cfg

### D-NUCA, 8MB, 16x8, 70nm technology

# L2 cache configuration
-nuca:row       16
-nuca:col       8
-cache:define   L2_row0:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row1:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row2:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row3:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row4:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row5:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row6:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row7:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row8:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row9:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row10:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row11:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row12:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row13:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row14:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define   L2_row15:1024:64:0:8:l:3:pipt:0:1:0:Membus
#8MB

# L1 cache configuration
-cache:define   L1_data:512:64:0:2:F:3:vipt:0:1:0:Onbus
-cache:define   L1_inst:512:64:0:2:l:1:vivt:0:1:0:Onbus
-cache:dcache   L1_data
```

```
-cache:icache   L1_inst

# Buses configuration
-bus:define
Onbus:16:1:0:0:16:0:L2_row0:L2_row1:L2_row2:L2_row3:L2_row4
:L2_row5:L2_row6:L2_row7:L2_row8:L2_row9:L2_row10:L2_row11:
L2_row12:L2_row13:L2_row14:L2_row15
-bus:define     Membus:16:4:0:0:1:0:SDRAM

# RAM configuration
-mem:define     SDRAM
-mem:clock_multiplier   27

# TLBs configuration
-tlb:define     TLB_data:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:define     TLB_inst:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:dtlb       TLB_data
-tlb:itlb       TLB_inst
```

Bibliografia

[1] C. Kim, D. Burger, S.W. Keckler: *An adaptive, non uniform cache structure for wire-delay dominated on-chip caches*, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Ottobre 2002.

[2] A. Desai, B. Metha, D. Sachdev, G. Muller: *Non uniform cache architectures for wire delay dominated caches*, CS/ECE 752 – Advanced Computer Architecture, 2003

[3] H. Kobayashi, I. Kotera, H. Takizawa: *Locality analisys to control dynamically way-adaptable caches*. SIGARCH Comput. Archit. News 33, 3 Giugno 2005

[4] S. Grechi: *Progettazione logica e definizione di protocolli e politiche per le cache D-NUCA triangolari*, tesi di laurea, A.A. 2002/2003.

[5] G. Diodato, R. Tolaro: *Metodologie di progettazione per Dynamic e Triangular-Dinamyc NUCA caches*, tesi di laurea, A.A. 2004/2005

[6] G. Gabrielli: *Applicazione della tecnica di riconfigurazione way-adapting alla progettazione di cache D-NUCA*, tesi di laurea, A.A. 2005/2006

[7] D. Di Mari: *Tuning di cache NUCA way-adaptable*, tesi di laurea, A.A. 2006/2007

[8] A. Bardine, P. Foglia, G. Gabrielli, C. A. Prete, P. Stenström: *Improving Power Efficiency of D-NUCA Caches*, Members of HIPEAC EU Network of Excellence, 2006

[9] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, C.A. Prete, P. Stenstrom: *Leveraging Data Promotion for Low Power D-NUCA Caches*, 11th EUROMICRO Conference on Digital System Design, Parma, Italy, 2008.

[10] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, C.A. Prete: *Implementation Issues of Way Adaptable D-NUCA Caches*, ACACES2008, L'Aquila, Italy, 2008

[11] A. Bardine, P. Foglia, G. Gabrielli, C.A. Prete, P. Strenstrom: *A Micro-Architectural Power-Saving Technique for D-NUCA caches, UCAS-4. Austin, TX, 2008.*

[12] A. Bardine, P. Foglia, C.A. Prete: *Way Adaptable D-NUCA caches*, ACACES2006, L'Aquila, Italy, 2006.

[13] R. Desikan, D. Burger, S. W. Keckler, T. M. Austin: *Sim-alpha: a validated execution driven Alpha 21264 simulator*, Technical Report TR-01-23, Deparment of Computer Sciences, University of Texas at Austin, 2001.

[14] D. Burger, A. Kägi, M. S. Hrishikesh: *Memory hierarchy Extensions to the SimpleScalar Tool Set*, Technical Report TR9925, Deparment of Computer Sciences, University of Texas at Austin, 2001.

[15] Standard Performance Evaluation Coporation, http://www.spec.org.