UNIVERSITÀ DEGLI STUDI DI PISA

FACOLTÀ DI INGEGNERIA

TESI MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTAZIONE E ANALISI DI UNA MEMORIA PRINCIPALE AD ALTE PRESTAZIONI BASATA SU PCM E DI UNA CACHE REALIZZATA CON 3D-DRAM

CANDIDATO:
GIUSEPPE REGINA

RELATORI:
PROF. COSIMO ANTONIO PRETE
PROF. PIERFRANCESCO FOGLIA
PROF. KRISHNA KAVI

A. A. 2012/2013

Alla mia famiglia, che mi ha sempre supportato.

Ai miei amici, che mi hanno sempre sopportato.

ABSTRACT

Il *Memory Wall* è un problema molto sentito da qualche anno a questa parte nell'industria tecnologica dei microprocessori: al fine di ridurre gli effetti di tale fenomeno sono state teorizzate ed in alcuni casi realizzate nuove soluzioni che fanno uso di un'architettura tridimensionale per limitare la latenza delle memorie principali anche sfruttando la possibilità di impilarle direttamente al di sopra dell'unità centrale.

In questo documento verranno analizzate tre architetture avanzate di memoria costruite con tecnologie differenti (DRAM, PCM) ed organizzate come cache (CMM).

Si vedranno pregi e difetti di ciascuna di esse e sarà effettuata una disquisizione esaustiva al fine di fornire utili indicazioni alle aziende circa i punti di forza e debolezza delle soluzioni analizzate per rendere più efficienti i processi produttivi e divenire più competitivi nel mercato di riferimento.

Infine verrà analizzata con maggior dettaglio una delle tre architetture con riferimento a benchmark affermati ed emergenti per il testing di sistemi ad alte prestazioni.

INDICE

ELENCO DELLE TABELLE

ELENCO DEI GRAFICI

ELENCO DELLE IMMAGINI

ABBREVIAZIONI

DRAM	Dynamic Random Access Memory
PCM	Phase Change Memory
CMM	Cache Main Memory
3D	Tre Dimensioni, Three-dimensional
IC	Integrated Circuit
TSV	Through Silicon Via
D2D	Die To Die
L1	Level 1 [Cache]
L2	Level 2 [Cache]
L3	Level 3 [Cache]
LLC	Last Level Cache
MC	Memory Controller
MRQ	Memory Request Queue
FIFO	First In, First Out
FRTD	Flash Replacement Technology Drive
ASID	Application Specific IDentifier
VA	Virtual Address
PA	Physical Address
TLB	Translation Lookaside Buffer
SSD	Solid State Disk / Solid State Drive
SRAM	Static Random Access Memory
LRU	Least Recent Used
PCM-MM	Phase Change Memory as Main Memory
3DLLC	3D DRAM as Last Level Cache
FTL	Flash Translation Layer
SCM	Storage Class Memory

SCSI	Small Computer System Interface
SATA	Serial ATA (Serial AT Attachment)
PMM	[3D-DRAM as] Part of Main Memory
CMOS	Complementary MOS
MOS	Metal–Oxide–Semiconductor
NMOS	N-Type MOS
PMOS	P-Type MOS
MESI	Modified Exclusive Shared Invalid [Protocol]
CAM	Content Addressable Memory
NUCA	Non Uniform Memory Access
UCA	Uniform Memory Access
SPEC	Standard Performance Evaluation Corporation
OLTP	On-Line Transaction Processing
HPC	High Performance Computing
AI	Artificial Intelligence
DDR3	Double Data Rate Type Three
API	Application Programming Interface
JIT	Just In Time [Compiler|Debugger]
IL	Intermediate-Language

AVVERTENZA

Il documento che segue è parte integrante di un progetto a più grande e larga portata che ha coinvolto, nei primi quattro mesi dell'Anno Domini 2013, tre autori differenti e che, separatamente, hanno sviluppato ciascuno una branca del macro progetto cofinanziato da AMD ed effettuato presso l'*University of North Texas*.

Ciò nonostante, come è naturale che sia, nel lavoro che coinvolge più persone e che parte da uno snodo comune, è inevitabile che alcuni argomenti riportati in questo documento, in (1) e (2) risultino molto simili sia per struttura che per contenuto: il macro progetto infatti aveva lo scopo di analizzare diverse soluzioni aventi protagonisti differenti modi di utilizzare la memoria 3D.

Il titolo del documento e le diverse citazioni all'interno del testo aiuteranno il lettore a comprendere quale parte sia stata sviluppata da un autore piuttosto che da un altro.

INTRODUZIONE

La sempre maggiore velocità dei processori, spesso esplicata tramite l'empirica legge di Moore (3), sebbene da un lato sia una spinta verso un maggior numero di istruzioni per unità di tempo, dall'altro essa accentua sempre più un problema noto in letteratura come *Memory Wall* (4) ossia la differente velocità di evoluzione dei processori rispetto a quella delle memorie che porta le memorie stessa ad essere ordini di grandezza più lente.

Allo scopo di ridurre tale problema si è pensato di costruire le memorie, invece che su un chip separato, sullo stesso chip dell'unità centrale affidandosi ad un'architettura di tipo tridimensionale anziché planare.

L'uso di tali tipi di memorie tuttavia ha solo ridotto il problema senza tuttavia eliminarlo definitivamente. Lo scopo di questo documento è mostrare una serie di nuove soluzioni hardware che siano in grado di attaccare, indebolire ed eventualmente sconfiggere il problema del Memory Wall almeno nella presente e prossima generazione.

Approccio tridimensionale: 3D Die Stacking

Con il termine *3D Die Stacking* si intende il nuovo ed interessante tipo di tecnologia, dianzi accennato, che permette di incrementare la densità di transistori mediante l'integrazione verticale di due o più die i quali comunicano tra di loro attraverso un'interfaccia ad alta densità e velocità. Ad esempio blocchi operazionali all'interno di un microprocessore possono essere sovrapposti al fine di ridurre al minimo la latenza di comunicazione tra gli stessi.

Una ulteriore, e sicuramente più interessante, proprietà di questo tipo di tecnologia risiede nella possibilità di realizzare i diversi die con diversi processi costruttivi e dunque consente di ridurre drasticamente i vincoli di progettazione per un processore, ottenendo così importanti miglioramenti in termini di consumo energetico e performance.

Nonostante le entusiasmanti promesse di questo nuovo tipo di tecnologia, è necessario essere cauti e considerare ogni aspetto del sistema: è chiaro che dissipare calore da due diversi strati di silicio attivo può comportare problemi di non facile o immediata soluzione.

Struttura tridimensionale

Nell'ambito di un circuito integrato (Integrated Circuit, IC) le piste metalliche sono una fonte primaria di overhead in termini di latenza, area ed energia consumata. Diversi studi (5) dimostrano che nelle varie connessioni all'interno di un microprocessore le piste metalliche utilizzate per le stesse possono partecipare fino ad una percentuale del 30% del consumo totale. In questo senso la tecnologia 3D Die Stacking consente di abbattere la lunghezza di tali piste introducendo connessioni verticali tra i vari dies, dette Through Silicon Via (TSV).

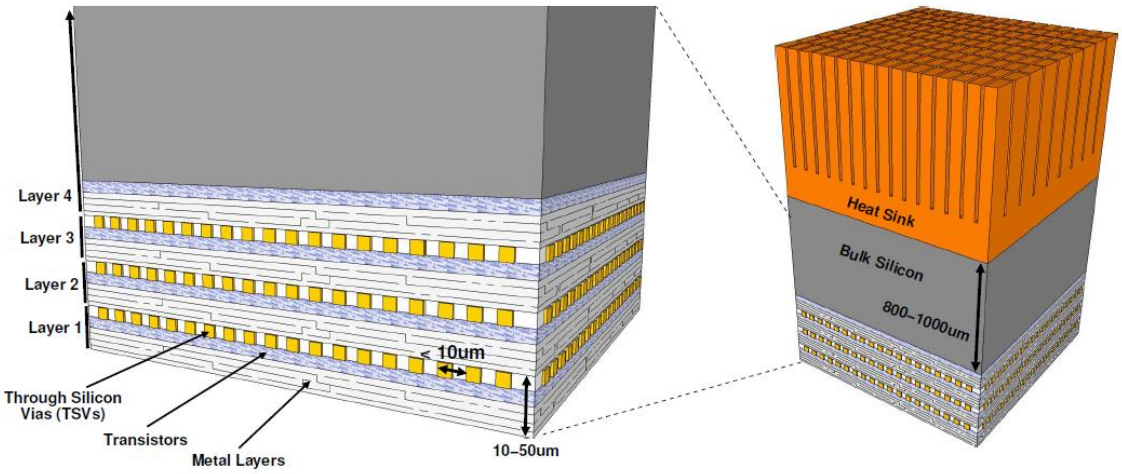

Figura : Struttura 3D di due die sovrapposti (tratta da: (8))

Vi sono diverse metodologie per la sovrapposizione di più die, tra i quali ricordiamo il wafer-to-wafer bonding (6), die-to-die bonding (7) e die-to-wafer bonding con diversi tipi di sovrapposizione.

3D Stacked DRAM

La tecnologia 3D Stacked DRAM si è presentata alla comunità scientifica che si occupa di Computer Architecture come una strada percorribile e promettente per ovviare al ben noto problema del gap di velocità tra la memoria principale e il processore,

conosciuto anche come *Memory Wall Problem*: attraverso l'uso di die di DRAM ad alta capacità sovrapposti al processore e l'uso massiccio di interconnessioni d2d ad alta velocità, l'integrazione tra queste componenti ha dato luogo ad una drastica diminuzione della latenza d'accesso alla memoria ed all'aumento della larghezza di banda nelle comunicazioni tra processore e memoria. Inoltre numerosi studi (8; 9) hanno dimostrato che la sovrapposizione della memoria DRAM sul processore porta notevoli miglioramenti anche in termini di *perfomance* e consumo energetico sebbene questi studi abbiano sempre presupposto un'organizzazione standard della memoria DRAM, ovvero prendendo come riferimento l'organizzazione classica delle attuali memorie planari.

Un'organizzazione più aggressiva per le memorie è stata proposta da Loh (9) e verrà esposta nei paragrafi successivi.

Cenni all'organizzazione e funzionamento di una classica memoria 2D DRAM

Una memoria DRAM è semplicemente una matrice di celle da un bit ciascuna delle quali implementata tramite un unico transistor con della logica accessoria necessaria ad accedere tali bit. La figura successiva illustra una generica gerarchia di memoria.

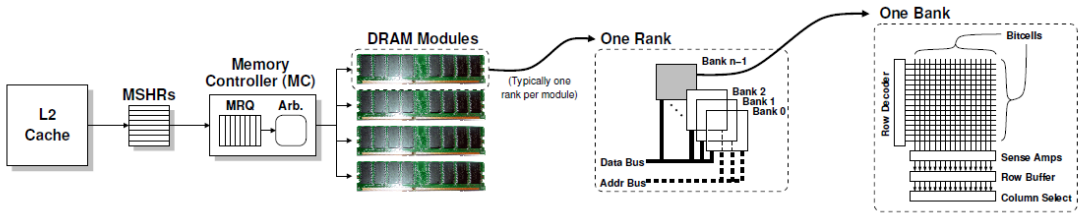

Figura : Organizzazione di una generica memoria RAM (tratta da (8))

Partendo dalla memoria cache di secondo livello, nel seguito denominata indifferentemente "L2" o "Cache L2", si nota come una miss in questo punto della gerarchia richiede un accesso alla memoria principale per soddisfare la richiesta. La logica della L2 inoltra la richiesta al controllore della memoria (*Memory Controller*, MC) il quale è deputato alla comunicazione con i chip di memoria DRAM. Tale richiesta può essere messa in una coda (*Memory Request Queue*, MRQ), la quale può esser gestita a piacimento (anche se la politica più utilizzata è quella FIFO) da un arbitro o da uno schedulatore.

A livello più alto una DRAM è dapprima divisa in ranghi o *ranks*, tipicamente uno o due per modulo. All'interno di ciascun rango la memoria è suddivisa in banchi. Ogni banco è, infine, una matrice di celle di bit, generalmente quadrata. Durante un'operazione di lettura in DRAM parti differenti dell'indirizzo fisico relativamente all'operazione stessa vengono utilizzate per selezionare rango, banco e riga. Un insieme di *sense amplifiers* legge il contenuto di una riga registrandone il contenuto in un *row buffer*: ciascun accesso sequenziale alla stessa riga può con essi risparmiarsi la lettura della riga stessa, avendo a disposizione il dato richiesto all'interno del *row buffer*. I pochi bit rimanenti dell'indirizzo selezionano infine la colonna all'interno del banco e il dato viene finalmente inviato all'ultimo stadio della gerarchia di memorie cache dal *memory controller*.

È altresì importante notare che le operazioni di lettura in DRAM sono distruttive, nel senso che dopo una lettura il dato ha bisogno di essere riscritto nella rispettiva riga. Ciò è dovuto al fatto che il dato viene letto scaricando i condensatori che memorizzano i vari bit attraverso una certa carica elettrica. Durante l'operazione di lettura la riga selezionata attiva i transistori di passo che scaricano la carica elettrica contenuta nelle celle di bit. I *sense amplifiers* registrano le variazioni di corrente sulla *bitline* e forniscono in uscita il dato memorizzato.

Occorre ricordare, poi, come il contenuto della DRAM vada periodicamente rinfrescato (operazioni di *refresh*) a causa delle inevitabili perdite di carica all'interno della matrice di bit.

Per aumentare le performance di questo tipo di memoria si va in genere ad incrementare il numero di ranghi o banchi aumentando il parallelismo; tutte queste componenti però possono sfruttate solo entro certi limiti: il numero di ranghi, ad esempio, è limitato dalla necessità di tenerne il più basso numero possibile in ogni modulo di DRAM mentre il numero di banchi è limitato dall'area richiesta per l'implementazione del banco stesso.

Cenni all'organizzazione e funzionamento di una 3D-DRAM

Nel seguito verranno presentate due differenti approcci per la costruzione di un'architettura di memorie 3D-DRAM: *Wide* e *True-3D*.

Approccio Wide-3D

La figura seguente mostra l'organizzazione di una DRAM tridimensionale progettata secondo quella denominata architettura *Wide-3D.*

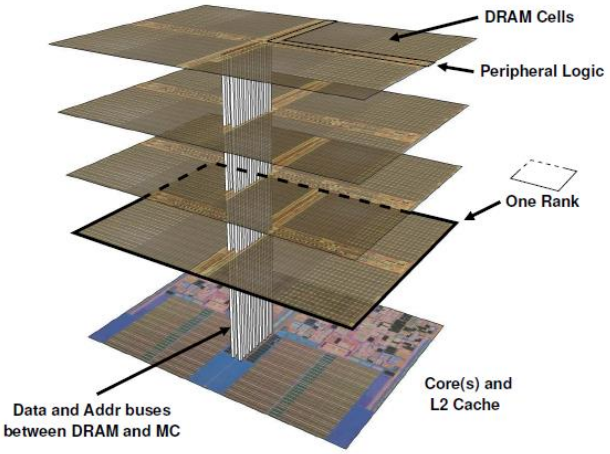

Figura : Organizzazione di un'architettura 3D wide (tratta da (9))

Tale nome deriva dal fatto che, così come il *layer* 0 è sostanzialmente equivalente alle odierne CPU, gli $N-1$ layer superiori contengono sia logica di controllo ed amplificazione come row buffers e *sense amplifiers* sia le vere e proprie matrici di bit. Il grande bus centrale è condiviso tra i vari *layer*, ciascuno implementante un rango della memoria e risulta essere un collo di bottiglia per transazioni parallele. È semplice comprendere dunque che la soluzione proposta da questa architettura, che poi è l'unica ad essere stata effettivamente realizzata, altro non è che la riproposizione, in miniatura e in formato 3D, dei vari moduli/ranghi/banchi presenti nell'approccio planare e del quale si è discusso in precedenza.

Approccio True-3D

La figura seguente mostra l'organizzazione di una *True-3D* DRAM. Gli $N-2$ *layer* superiori contengono soltanto le matrici di bit e sono realizzati con processi tecnologici ottimizzati per la densità. Già qui si nota una grande differenza con l'approccio classico che, dovendo mantenere sullo stesso *layer* componenti differenti, necessita di avere un approccio più conservativo. La tridimensionalità inoltre è sfruttata intensamente anche sul generico *layer*: come si vedrà a breve i ranghi sono estesi su più *layer* e non sono più planari e questo ha come diretta conseguenza l'avere uno o più

banchi di memoria appartenenti ad un certo rango concentrati in una frazione di *layer*. Questi banchi a loro volta utilizzano la terza dimensione in una maniera che figuratamente ricorda una serie di edifici (i banchi) costruiti su una piazza (*layer*) a sua volta sviluppata come un parcheggio a più piani (l'intero *stack* 3D).

Il *layer* 1 contiene la logica di controllo come *row decoders*, *sense amplifiers*, *row buffers* eccetera. Un intelligente posizionamento dei componenti permette di suddividere il grande bus centrale in vari bus più piccoli, ciascuno dei quali è relativo ad un singolo rango tridimensionale: nell'organizzazione *True-3D* il singolo rango è riorganizzato su diversi *layer* in modo da ridurre in lunghezza il percorso del dato e conseguentemente aumentare la frequenza massima di funzionamento della memoria mentre il bus centrale scompare e lascia il posto a numerose TSV che, indipendentemente, possono gestire richieste relative a ranghi differenti.

Il *layer* 0 infine è il *layer* contenente la CPU vera e propria e non si discosta molto dal corrispettivo *layer* 0 dell'approccio classico.

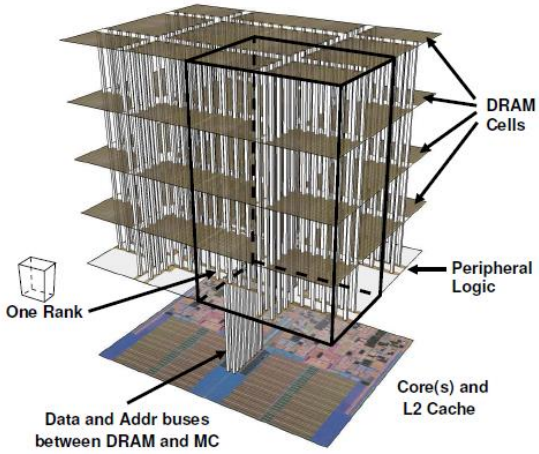

Figura : Organizzazione di un'architettura True-3D (tratta da (9))

Oltre al riposizionamento dei componenti, la separazione tra *layer* logico e *layer* di memoria comporta un altro importante beneficio. L'isolare la logica di controllo dal resto della memoria consente, infatti, di utilizzare diversi processi tecnologici nello stesso package: le celle di memoria sono implementate usando logica NMOS tradizionale, caratterizzata da alta densità mentre il *layer* logico è implementato in logica CMOS, caratterizzata dall'alta velocità.

Phase Change Memories

Lo *scaling* del processo produttivo nella fabbricazione di memorie, obbiettivo di primaria importanza nell'industria microelettronica, è attualmente in pericolo in quanto i meccanismi di *sensing* e immagazzinamento dei dati diventano sempre meno adattabili, soprattutto nelle tecnologie più diffuse come ad esempio nelle memorie DRAM.

In contrasto con quanto appena esposto, la tecnologia PCM (*Phase Change Memory*) fornisce un meccanismo di immagazzinamento dati non-volatile soggetto ad una forte scalabilità: in tale tecnologia durante un'operazione di scrittura un transistor d'accesso inietta corrente nel materiale di immagazzinamento e introduce termicamente un cambiamento di fase, il quale viene rilevato durante le letture. La PCM dunque, basandosi su correnti analogiche ed effetti termici, non necessita di immagazzinare cariche elettriche. In quest'ottica, la corrente di programmazione, necessaria al cambiamento di fase nel materiale costituente la PCM, scala linearmente. Tale meccanismo di *scaling* è stato dimostrato in un prototipo di dispositivo facente uso di una tecnologia a 20 nm. (10) (11)

La PCM comporta comunque anche diversi inconvenienti, il più evidente dei quali riguarda le latenze d'accesso che, nonostante siano di alcune decine di nanosecondi, sono diverse volte più grandi di quelle relative ad una normale DRAM. Allo stato attuale dell'arte le operazioni di scrittura richiedono una intensa iniezione di corrente, il che implica una grande energia dissipata. Inoltre le operazioni di scrittura, in quanto ad alta corrente, inducono espansioni e contrazioni termiche all'interno del materiale, degradandone l'integrità e limitandone la durata a centinaia di milioni di scritture.

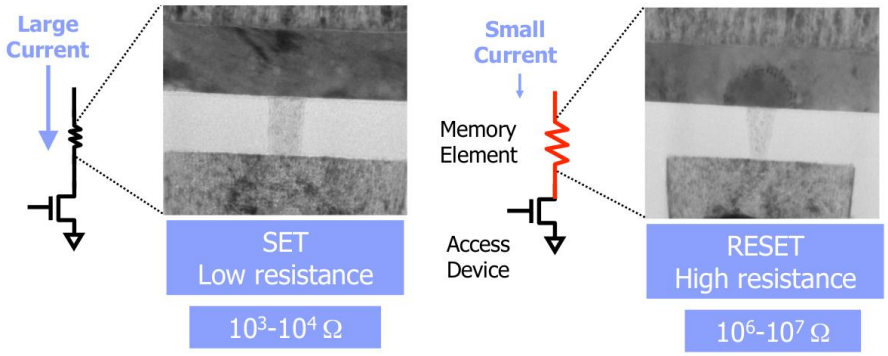

Figura : Visione al microscopio delle due fasi del materiale calcogeno (tratta da (12))

Cella di memoria di una PCM

La cella elementare di memoria nella tecnologia PCM è costituita da due elettrodi separati da una resistenza e da un materiale a cambiamento di fase, tipicamente un materiale calcogeno (Figura). Il materiale calcogeno presenta particolari caratteristiche, che lo rendono adatto all'utilizzo nell'ambito delle memorie; esso presenta infatti una grande differenza di resistività tra la fase policristallina e la fase amorfa. Quando il materiale viene depositato all'interno del circuito si presenta in fase amorfa, con un impedenza tipica nell'ordine dei $10^7 \Omega/sq$. Quando riscaldato il materiale presenta una diminuzione di impedenza che è funzione della temperatura; si osserva che tale diminuzione è di un fattore $\sim$100 intorno ai 150°C, mentre è di un fattore $\sim$10 intorno ai 350°C.

Riscaldando dunque il materiale, facendolo attraversare da una corrente controllata con forma d'onda regolare, è possibile riscaldarlo fino alla temperatura voluta ed ottenere conseguentemente un grado di impedenza noto che provvederà dunque a memorizzare il dato: infatti durante una lettura si misura la tensione ai capi della cella di memoria attraversata da una corrente.

Nei vari casi di alta o bassa impedenza la caduta di tensione ai capi della cella sarà diversa e può essere sfruttata per la memorizzazione del dato e la distinzione tra 0 e 1.

Inoltre, poiché l'impedenza del materiale calcogeno può essere variata su diversi valori abbastanza diversi tra loro è possibile memorizzare fino a 4 bit in una singola cella di memoria, aumentando dunque la densità di memorizzazione di un fattore 4.

Ad ogni modo, la cella di memoria è costituita da un sottile film di materiale a cambiamento di fase il quale è a contatto su entrambi i lati con elettrodi metallici. La transizione tra le varie fasi del materiale può essere indotta in una scala temporale che va dai $5ns$ ai $500ns$ attraverso impulsi elettrici controllati che causano un riscaldamento del materiale per effetto Joule.

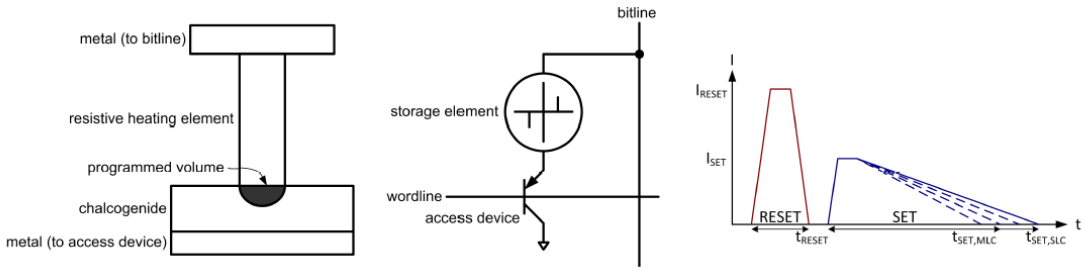

Figura : Cella elementare di memoria e tempi delle operazioni (tratta da (12))

Supponiamo per semplicità e comunque senza ledere alla generalità della trattazione, di avere una cella che supporti solamente due possibili stati: alta impedenza e bassa impedenza. Per programmare una cella in stato di alta impedenza, operazione di RESET, si applica un impulso elettrico al fine di alzare la temperatura di buona parte del materiale calcogeno poco al di là della sua temperatura di fusione (~660°C). La densità di corrente richiesta per quest'operazione è nell'ordine dei $\frac{5 \div 20MA}{cm2}$. Dal momento che la costante di tempo termica di una cella costruita con tecnologia inferiore ai $90nm$ è minore di $10ns$, le condizioni di stato termico stabile possono essere ottenute in scale di tempo simili mentre il volume fuso può essere portato nello stato amorfo di alta resistività se il tempo di discesa dell'impulso di RESET avviene nel giro in pochi nanosecondi.

La riprogrammazione della cella allo stato di bassa impedenza anch'essa basata su effetto Joule. Per settare una cella (operazione di SET) viene applicato un impulso elettrico tale da rendere lo strato amorfo conduttore. La corrente che scorre all'interno del dispositivo è controllata al fine di portare il materiale amorfo alla temperatura di cristallizzazione.

I tempi necessari al completamento delle operazioni di SET e RESET sono riportati in Figura . (12)

ARCHITETTURE PROPOSTE

Partendo dunque dalle tecnologie nella loro forma base, ossia 3D-DRAM nella versione *True-3D* e PCM come memoria principale è possibile espanderle creando architetture del tutto nuove ed innovative anche coinvolgendo tecnologie che non appartengono al gruppo di quelle emergenti.

In particolare si proporranno tre architetture che fanno uso, in maniera più o meno accentuata, sia delle memorie 3D sia delle *Phase Change Memories*; tali memorie sono la *Cache Main Memory* (13) (2), 3D-DRAM come *Last Level Cache,* 3D-LLC e 3D-DRAM come parte della memoria principale (1). L'architettura che verrà poi analizzata in maniera specifica e che includerà le simulazioni in questo documento è: "Phase Change Memory e 3D-DRAM come LLC"

Cache Main Memory

Il lavoro effettuato sulla *Cache Main Memory*, in seguito denominata per brevità CMM, è un lavoro di espansione rispetto a quello eseguito nei precedenti anni da (13). Tale precedente studio infatti privilegiava aspetti relativi alla velocità di esecuzione e risposta di un sistema dotato di CMM mentre lo studio qui presentato parte dai risultati già raggiunti e li espande fornendo anche quantità di energia/potenza dissipata per ogni singolo componente della memoria nonché numerose altre migliorie che verranno evidenziate ed esplicate nella narrazione seguente.

Organizzazione della Memoria CMM

L'idea di fondo su cui si basa la CMM è la vicinanza fisica della memoria 3D al processore: questo permette di vedere la memoria principale sia come cache sia come memoria principale. Ecco dunque che già si delinea una prima caratteristica della CMM che poi è anche quella più importante nonché quella che le ha donato il nome: la dualità

della memoria che, a seconda dei punti di vista, ossia a seconda dell'operazione da effettuare, può essere interpretata sia come cache sia come memoria principale.

Comprendendo dunque, oltre alla memoria principale anche le cache di primo e secondo livello oltre allo *storage* secondario è possibile ricavare la struttura modulare del sistema mostrata nella figura seguente.

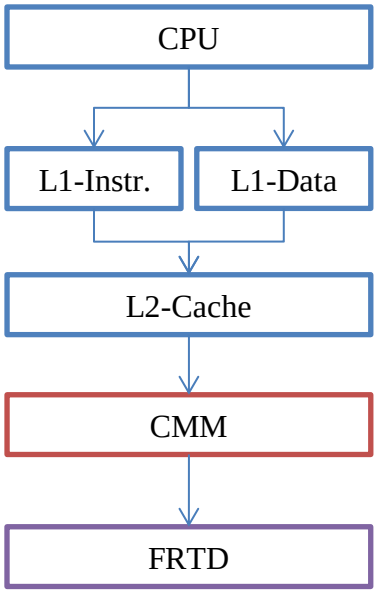

Figura : Organizzazione modulare del sistema CMM

I moduli CPU, Cache L1 e Cache L2 sono dei moduli ovviamente classici ossia analoghi a quelli presenti nei sistemi attuali; i moduli CMM e FRTD (*Flash Replacement Technology Drive*) sono stati invece appositamente progettati e simulati per soddisfare specifiche direttive.

Cache di primo e secondo livello

Per iniziare è innanzitutto necessario specificare che entrambi i livelli di cache classici sono stati modificati per funzionare con un indirizzamento virtuale. Infatti una delle specifiche del sistema CMM + FRTD è quella di dover essere abbastanza veloce da non dover richiedere al *layer* software del sistema operativo alcun intervento nel caso vi sia un *page fault*. L'indirizzamento virtuale avviene dunque utilizzando come *tag* la coppia $\langle Asid, VA \rangle$: tale approccio sebbene porti un *overhead* in termini di area utilizzata,

velocità di accesso e potenza consumata, risulta essere un buon compromesso poiché evita, cosa che invece accade nei sistemi moderni attuali come l'Intel® i7 Series™ (14), il *flush* completo delle cache dopo un cambio di contesto a seguito di un evento di page fault. Inoltre, poiché i primi due livelli di cache sono inclusivi e pertanto sono tra loro sincronizzati sul contenuto, si dovrà non solo effettuare il *flush* sulla cache primo livello ma anche su quella di secondo livello. Una singola linea di cache di primo o secondo livello si presenta dunque nel modo seguente:

Tabella : Singola linea di Cache L1/L2

ASID **16 bit**	Virtual Tag	Data Line **128 byte**

L'ASID, che può essere visto senza perdita di generalità come il PID del processo che ha richiesto/effettuato l'operazione di memoria, è su 16 bit ricordando come effettivamente, sui sistemi Linux, i PID dei processi sono compresi tra 0 e 32768 ossia rappresentabili come intero senza segno a 16 bit (15). Si noti come tale architettura non è invece adatta a sistemi Windows® in quanto i PID su tale sistema sono multipli di 4 e di tipo `DWORD` (16) sebbene la compatibilità si possa semplicemente ottenere portando a 32 i bit riservati all'ASID.

Componentistica della CMM

Il modulo CMM, poiché le cache di livello inferiori sono indirizzate virtualmente, riceve direttamente indirizzi virtuali. È pertanto necessaria una traduzione da indirizzo virtuale a indirizzo fisico che prenda un quantitativo di tempo nettamente inferiore all'attuale mezzo di traduzione per mezzo del *page walking*.

TLB

Al fine di velocizzare quanto più possibile l'operazione di traduzione la CMM è dotata di un TLB che associa, in maniera non dissimile dalle cache, alla coppia $\langle Asid, VA \rangle$ l'indirizzo fisico del dato richiesto. (La cache ovviamente associa, all'indirizzo virtuale, il dato stesso)

Tabella : Entrata del TLB

ASID **16 bit**	Virtual Tag	Utility Bits	Page Physical Address **20 bit**

La tabella precedente mostra una generica entrata del TLB discusso: si noti come la struttura è pressoché identica a quella di una linea di cache con l'aggiunta dei bit di utilità come bit di validità di presenza e di modifica e con la metamorfosi del campo dati da un valore di memoria a un indirizzo della stessa.

Una miglioria apportata al TLB da questo studio, che non era stata presa in considerazione precedentemente (13), è la sua espansione da un piccolo numero di entrate e da un'associatività completa ad una versione più capiente ed associativa ad insiemi. I capitoli di questo documento dedicati alle simulazioni mostreranno da quali scelte si è partiti e quale di esse è risultata essere la migliore in termini di tempo di risposta, di potenza consumata e area occupata.

Studi precedenti (13) hanno inoltre identificato nel classico meccanismo di paginazione un ulteriore miglioramento possibile: poiché i dischi tradizionali iniziano ad essere gradualmente sostituiti da dischi a stato solido i cui tempi di risposta sono paragonabili a quelli di una memoria di qualche anno fa, è davvero necessario trasferire dati pari alla dimensione di una pagina nel momento in cui si ottiene *page fault*? La risposta è negativa e pertanto si è effettuata la scelta di suddividere la memoria in grandi pagine da 32KB ciascuna e di suddividere a loro volta tali pagine in piccole sottopagine da 128 bytes ossia della stessa dimensione di una linea di cache.

Il campo *Physical Address* del TLB dunque può essere ristretto dai 64 bit dei sistemi tradizionali ad un campo più piccolo a 20 bit che altri non è che l'indice della pagina fisica all'interno della memoria 3D: il numero di bit è calcolato in funzione della dimensione massima della memoria e, nei casi di studio qui riportati, varia dal numero di bit necessari a rappresentare 262144 entrate fino al numero di bit necessari a rappresentarne 1048576 ossia è sicuramente minore o al più uguale a $\log_2 1048576 = 20$. Questo riduce di un 29% la grandezza di una singola linea di TLB con conseguente risparmio teorico di potenza e area occupata.

Bitmap

Alla richiesta in memoria di un dato non ancora presente in memoria si ha, quindi, un page fault che porterà in memoria fisica solo la sottopagina relativa al dato richiesto e memorizzerà nel TLB un'entrata relativa alla pagina madre che la conteneva. Per definire quindi quante e quali delle sottopagine relative ad una pagina sono in un certo momento presenti in memoria fisica è necessario espandere il TLB con una struttura chiamata *bitmap*. Per ogni entrata del TLB tale *bitmap* sarà composta da $256n$ bit dove n rappresenta il numero di informazioni da memorizzare per ogni sottopagina. In ogni pagina da 32KB infatti sono presenti 256 sottopagine da 128 byte l'una. Per ognuna di queste sottopagine bisogna memorizzare per lo meno il bit di presenza ma possono essere una buona scelta anche il bit di validità e di modifica onde evitare inutili riscritture su un back storage come l'SSD che ha un numero di riscritture per settore molto limitato rispetto a quello di un hard disk tradizionale. La scelta finale ricade in 2 bit riservati alla validità/presenza e alla modifica pertanto in definitiva una linea di TLB sarà composta nella seguente maniera:

Tabella : Entrata del TLB comprendente la bitmap

ASID **16 bit**	Virtual Tag	Utility Bits	Page Physical Address **20 bit**	Bitmap **512 bit**

SRAM

Supponendo che il dato richiesto non sia nel TLB ciò che accade nei sistemi tradizionali è il *page walking* che, pur essendo eseguito in hardware, può richiedere del tempo soprattutto se l'albero delle tabelle delle pagine da scorrere è su un numero di livelli superiore a 3 come accade nei sistemi che indirizzano una memoria virtuale superiore a 4 GB.

L'architettura CMM invece fa intervenire, nel caso di una *TLB-miss*, il meccanismo *cache-like* di traduzione dell'indirizzo che, grazie anche all'indicizzazione a insieme, velocizza enormemente l'operazione di traduzione. Prima di passare alla descrizione dell'architettura interna della cache della CMM è necessario esporre da quali campi si considerano essere formati gli indirizzi virtuali e fisici relativi alla CMM.

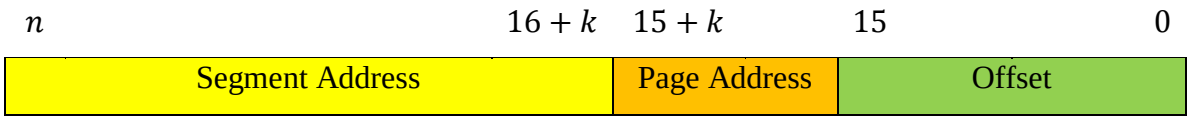

Tabella : Struttura di un indirizzo virtuale

Supponendo che ogni pagina sia grande 32 KB si può facilmente riservare, senza necessità di traduzione come d'altronde avviene nei sistemi moderni, un offset a 15 bit nell'indirizzo virtuale.

Un'ulteriore decisione progettuale, che non era stata prevista in (13), è stata quella di dividere la memoria virtuale in segmenti ognuno dei quali capaci di contenere k pagine distinte. Ovviamente risulta che il numero di bit riservati all'indirizzo di segmento è pari a $n-15-k$. Valori reali di progetto per n e k sono 64 e 10: vi sono dunque $2^{10}=1024$ pagine virtuali per ogni segmento virtuale.

Per passare all'indirizzo fisico è dunque necessario comprimere l'indirizzo virtuale da $n=64$ a m bit dove il valore di m è uno tra $\{33,34,35\}$ in funzione della memoria considerata che, in questo documento oscilla tra 8 e 32 GB.

Si è scelto dunque di dividere anche la memoria fisica in segmenti ciascuno dei quali appartenente in un certo istante ad un solo processo e ognuno dei quali composto da $h \leq k$ pagine in maniera da ottenere competizione nella scelta degli indirizzi. La scelta progettuale è stata quella di scegliere $h=6$ e pertanto si hanno 1024 pagine virtuali che competono in uno spazio di 64 pagine fisiche. Il numero di bit riservati al segmento è

conseguenza di questa suddivisione e pertanto oscilla da $s = 12$ a $s = 14$ a seconda della dimensione totale della memoria.

La scelta semplificativa effettuata in una prima analisi dagli autori di (13) e ripetuta in quanto non pertinente con lo studio desiderato dall'autore di questo documento e dei suoi fratelli letterari (2) e (1) è stata quella di presupporre un agente esterno, eventualmente software, che si occupi della veloce traduzione mediante tradizionale *page walking* dell'indirizzo di segmento. É interessante notare come tale traduzione, anche se effettuata via software, non richieda valori di tempo elevati in quanto il valore di output è solo di $12 \div 14$ bit il quale pertanto può essere ottenuto mediante un *page walking* ad un livello che degenera in una semplice *lookup table*.

Risulta quindi evidente che, presupponendo un agente esterno per la parte alta dell'indirizzo e presupponendo la non necessità di tradurre la parte bassa dell'indirizzo, ciò che rimane effettivamente da tradurre sono i bit della parte centrale tramite una compressione, usando i valori di progetto elencati precedentemente, da 10 a 6 bit.

La metodologia precedentemente utilizzata per effettuare tale traduzione era quella di creare, all'interno della CMM, una struttura dati hardware chiamata CMM TAG/Location la cui struttura è la seguente.

Tabella : Struttura TAG/Location utilizzata in (13)

Segment Address	Virtual TAG **10 bit**	Protection	Output Data **6 bit**

Il campo *Segment Address* rappresenta l'indirizzo fisico di segmento nel quale si trova il dato richiesto alla CMM. Il campo *Virtual TAG* è la porzione di 10 bit da comparare con quelli dell'indirizzo virtuale in arrivo dalla CPU e per il quale non è stato possibile effettuare traduzione via TLB.

Il campo *Protection* contiene bit relativi alla protezione della pagine ossia i bit `EXECUTE/NO-EXECUTE`, `READ/WRITE` e `USER/SUPERVISOR` il cui significato, per quanto noto, è per completezza riportato nell'elenco sottostante:

- `EXECUTE/NO-EXECUTE`: Indica se la pagina indicata dalla riga attuale è una pagina che contiene o meno codice eseguibile;
- `READ/WRITE`: Indica se la pagina è di sola lettura o meno;

- `USER/SUPERVISOR`: Indica se la pagina appartiene ad un processo che esegue in modalità utente o supervisore.

Il campo *Output Data* infine contiene i 6 bit relativi alla traduzione che andranno a formare la parte centrale dell'indirizzo fisico.

La miglioria apportata a questo approccio, forse un po' semplicistico, è stata quella di introdurre una coerenza tra SRAM e DRAM tale per cui una pagina inserita alla i −esima riga della SRAM risulta poi essere all'i −esima pagina fisica in memoria. Attraverso questo stratagemma la memorizzazione dei 6 bit non è più necessaria e il meccanismo di traduzione dell'indirizzo è quello rappresentato in figura seguente:

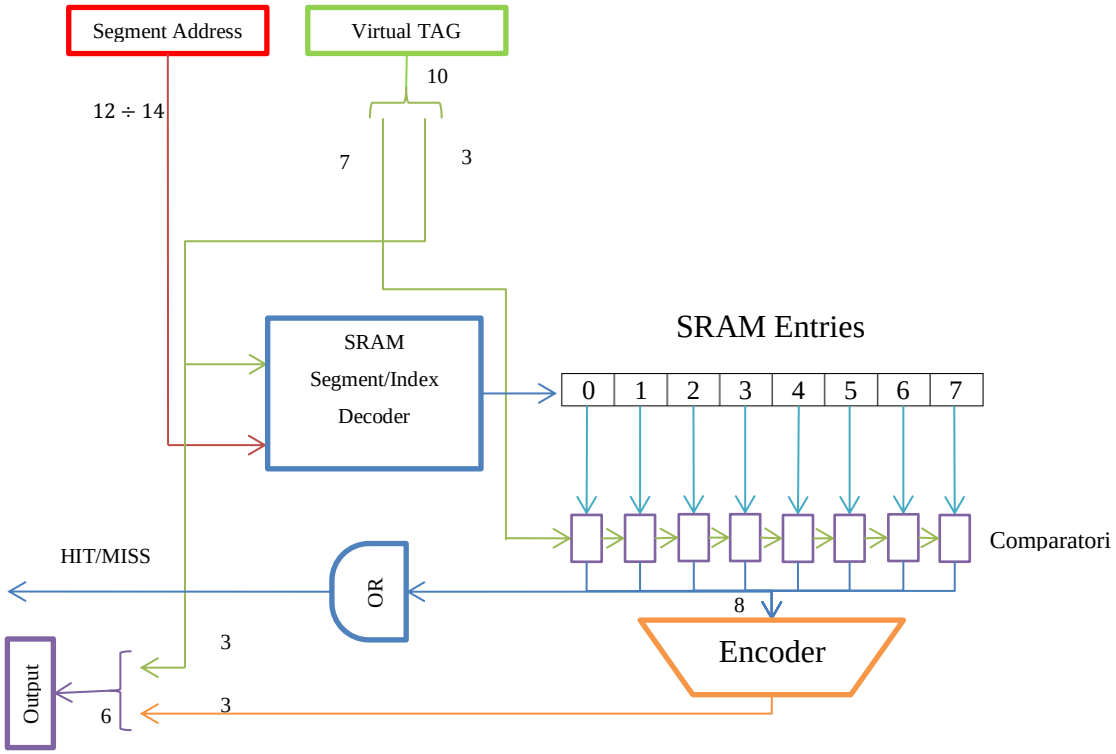

Figura : Struttura interna della SRAM

L'elenco dei passaggi effettuati è il seguente:

- Utilizzare l'indirizzo di segmento tradotto precedentemente assieme ai 3 bit meno significativi del TAG (*index*) per accedere al corretto set in SRAM;
- Effettuare 8 contemporanee comparazioni (in generale si dovrebbero effettuare tante operazioni quanta è l'associatività a insiemi della memoria;

tale scelta è basata su studi precedenti che hanno dimostrato che 8 −way è la migliore associatività possibile in un *range* da *direct-mapped* a 32)

- Mandare gli output della comparazione ad un *encoder*;

- Utilizzare l'uscita su 3 bit *dell'encoder* come parte bassa dell'output
- Utilizzare *l'index* su 3 bit come parte alta dell'output

Al termine delle operazioni ampiamente esplicate è quindi possibile presentare la struttura interna di un indirizzo fisico che si presenterà del tutto analoga a quella di un generico indirizzo virtuale eccezion fatta per la lunghezza dei vari campi; si ricorda che m potrà variare, in funzione della grandezza della memoria, da 33 bit per la versione a 8 GB fino a 35 bit per la versione a 32 GB.

m … 22	21 … 15	14 … 0
Segment Address	Page Address	Offset

Tabella : Struttura di un indirizzo fisico

DRAM

Una volta ottenuto l'indirizzo fisico, grazie al meccanismo innovativo di traduzione proposto, è necessario entrare effettivamente in memoria e caricare il dato all'interno delle cache di basso livello per accessi successivi più rapidi. L'indirizzo fisico ottenuto innanzitutto è un indirizzo di pagina. Poiché però, l'unità minima di trasferimento è la sottopagina, è necessario calcolare in quale sottopagina fisica cade l'indirizzo calcolato precedentemente: il calcolo è semplice una volta note le dimensioni di pagina e sottopagina; su 15 bit di offset se ne riservano 7 per l'offset di sottopagina e 8 per l'indice di sottopagina ricordando che ogni sottopagina è 128 byte. L'indirizzo fisico nella sua forma definitiva risulta quindi suddiviso come mostrato nella tabella seguente.

m … 22	21 … 15	14 … 6	5 … 0
Segment Address	Page Address	Subpage Index	Subpage Offset

Tabella : Struttura definitiva di un indirizzo fisico

Una volta acceduta la pagina è necessario controllare se la sottopagina richiesta è effettivamente presente in memoria oppure deve essere caricata dall'unità secondaria di

memoria. Tale operazione viene eseguita controllando la *bitmap* della pagina che, per pagine non residenti nel TLB, è memorizzata come intestazione all'interno della pagina fisica. Se dunque la sottopagina richiesta è in memoria allora la richiesta può essere soddisfatta e il dato può essere portato in cache L2. Se invece la sottopagina richiesta non è in memoria è necessario trasferirla dal FRTD e posizionarla nel giusto posto in memoria 3D oltre a dover aggiornare:

- una delle entrate della SRAM nel caso fosse la prima sottopagina a mancare (in questo caso si parla di *Page Miss*)
- il TLB inserendo la nuova entry (in caso di *Page Miss*) e marcando la *bitmap*.

Nel caso di trasferimento da FRTD, causa la maggiore velocità di risposta che si associa a quest'ultimo in comparazione con gli hard disk magnetici tradizionali, il processore entrerà in stallo attendendo il termine del trasferimento: non si effettuerà quindi alcun cambio di contesto che comporterebbe invalidazione di TLB e cache di basso livello.

Row Buffer

Per quanto riguarda l'accesso alla DRAM stessa un'aggiunta effettuata è stata quella relativa alla progettazione e implementazione di un modulo capace di simulare il comportamento della memoria dotata di *row buffer*.

Ricordando la Figura notiamo che un rango tridimensionale di memoria contiene, separati sui vari *layer*, un certo numero di banchi: tutti questi banchi sono collegati tramite TSV tra di loro (si vede molto bene in figura) e fanno quindi riferimento, al *layer* 1, ad una singola porzione di logica statica di controllo nella quale sono contenuti i *row buffers*. Organizzati come piccole cache tali *row buffers* velocizzano enormemente (17) le operazioni di ottenimento del dato se questi era stato precedentemente acceduto. Il numero di *row buffer* supposti per ogni gruppo di banchi organizzati su più *layer* in un singolo rango è 4.

Rimpiazzamento dei dati nella SRAM

Un ulteriore problema riguarda l'eventuale rimpiazzamento da effettuare in SRAM se tutto il set di destinazione risulta occupato. Per minimizzare i rimpiazzamenti inutili si è scelto di attuare una politica pseudo-LRU tramite un registro a scorrimento.

Quando dunque la SRAM produce un output di *miss* è necessario trasferire da FRTD la bitmap da posizionare nel TLB e la giusta sottopagina relativa al richiedente l'operazione. Anche in questo caso, data la presupposta velocità del sistema FRTD il processore non effettuerà alcun cambio di contesto ma attenderà in stallo che l'operazione sia terminata.

È quindi ora possibile presentare un diagramma di flusso che rappresenta tutte le operazioni effettuate dalla CMM dal momento di arrivo della richiesta al soddisfacimento della stessa.

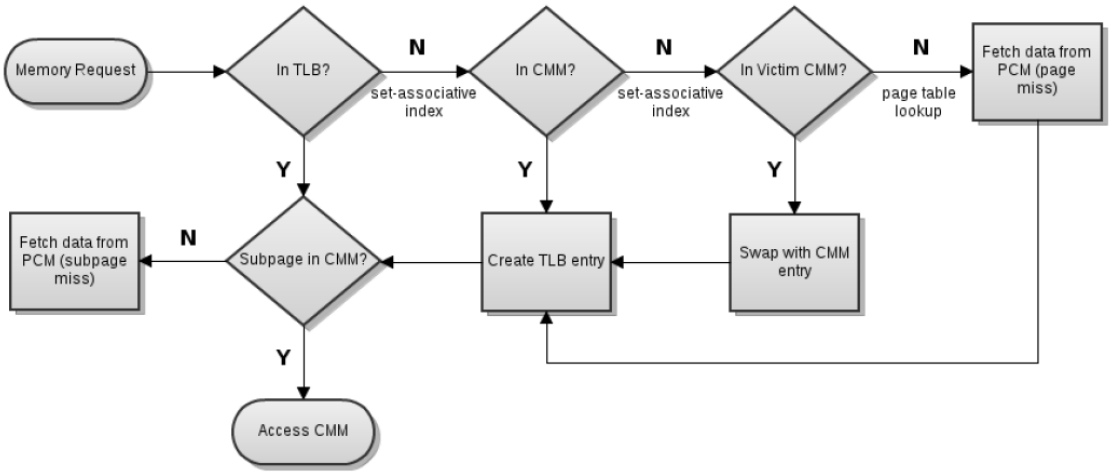

Figura : Algoritmo completo di accesso alla CMM (tratta da (13))

Victim Cache

La figura precedente mostra, tra le varie situazioni descritte, anche una situazione nuova relativa alla presenza o meno della pagina in una speciale locazione della CMM denominata *Victim Cache*. Tale zona di memoria non verrà trattata poiché non è stata né utilizzata né in alcun modo migliorata. L'uso di una *Victim Cache*, sulla quale sono presenti, sotto un punto di vista generale, molte informazioni in letteratura, in CMM

infatti non portava alcun beneficio visibile (13) e dunque è stata disattivata in tutte le simulazioni.

FRTD come Back Storage

Ultimo componente gerarchico nell'architettura della CMM è lo spazio di archiviazione di massa destinato allo *swap* delle pagine fisiche non più necessarie in memoria.

La memoria FRTD, che può essere pensata indifferentemente come un disco a stato solido (SSD) o una memoria a cambiamento di fase (PCM) ha come unica caratteristica richiesta quella di essere almeno $2 \div 3$ ordini di magnitudo più veloce, sia in lettura che in scrittura, rispetto ad un classico disco a rotazione magnetica: questa caratteristica è fondamentale per garantire il corretto funzionamento teorico del sistema. Con un disco magnetico infatti, sebbene il funzionamento continui ad essere garantito, il fatto di dover mettere in stallo il processore per un numero di cicli di clock pari alla durata dell'accesso al disco magnetico ossia un numero di cicli ricavabili come

$$\#cycles = accessTime * cpuFrequency$$

Ossia, per un processore a 3GHz e un disco a 15000rpm:

$$\#cycles = 2\ ms * 3GHz = 6000000$$

Risulta in un'immane perdita di prestazioni in quanto 6000000 è un numero di cicli di clock spropositato se si pensa che in un sistema con schedulazione dei processi Round-Robin lo slot assegnato a ciascun processo è pari a soli $50ns$.

Con una memoria veloce invece si ha che, anche con un solo ordine di magnitudo di maggiore velocità, il numero di cicli di clock effettivamente persi è limitato a 6000.

Sebbene questo ammontare di cicli di processore non sia del tutto trascurabile, si comprende bene come si riesca a conseguire una buona ottimizzazione del processo di cambio di contesto in caso di I/O da disco a causa di un page fault.

Phase Change Memory e 3D-DRAM come LLC

PCM come Main Memory (PCM-MM)

Il sempre crescente numero di processori presenti nei moderni sistemi ad alte prestazioni richiede che il sottosistema memoria scali bene in capacità al fine di poter contenere i *working set* dei programmi eseguiti su tali sistemi. Per diversi decenni la tecnologia DRAM è stata alla base delle memorie principali nei sistemi di elaborazione.

Tuttavia, con l'aumentare della dimensione della memoria, una porzione significativa della potenza totale e del costo di un sistema di elaborazione è stata dedicata alla realizzazione della memoria principale. Inoltre, come già detto, lo *scaling* della memoria è diventato un serio problema da affrontare; tutta questa combinazione di fattori conduce alla disdicevole conseguenza che la tecnologia DRAM scala ad un ritmo più lento rispetto alla tecnologia standard CMOS.

Se non affrontato prontamente questo problema potrà portare, nel prossimo futuro, ad avere sistemi limitati nelle performance da capacità di memoria non adeguate.

Ecco il motivo per cui i progettisti di sistemi stanno attualmente esplorando nuove tecnologie con cui costruire nuovi tipi di memorie, le quali possano garantire una maggiore capacità rispetto alla tecnologia DRAM, rimanendo sempre competitive in termini di *performance*, costi e consumo energetico. In tale ottica la tecnologia PCM si propone come valida soluzione.

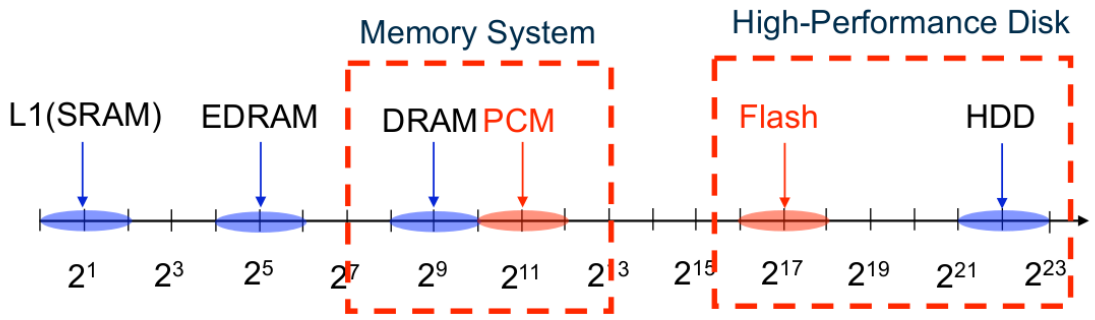

Figura : Latenze di accesso tipiche per varie tecnologie (tratta da (11))

La figura precedente mostra, in cicli di clock, latenze tipiche d'accesso per varie tecnologie di memoria e il loro rispettivo posizionamento nella gerarchia di memoria stessa. Una tecnologia più densa della DRAM e con tempi di accesso compresi tra quelli

della DRAM e quelli di un hard disk può colmare il *gap* tra hard disk e DRAM. Cache per hard disk basate su memorie Flash sono già state proposte per colmare parte di questo *gap* e ridurre il consumo energetico.

Essendo comunque le memorie Flash due o tre ordini di grandezza più lente delle normali DRAM, è comunque importante incrementare la capacità della memoria principale per ridurre gli accessi al disco (nonostante questo possa essere stato ottimizzato con memorie Flash). Le latenze più vicine a quelle di una DRAM rendono dunque la PCM un'attraente tecnologia per incrementare la capacità della memoria principale. Si consideri inoltre che le celle PCM hanno una durata media 1000 o più volte superiore a quella di celle Flash.

Organizzazione di una memoria principale realizzata tramite PCM

In modo del tutto simile all'organizzazione di una memoria convenzionale, le celle PCM possono essere organizzate in ranghi e banchi. Tuttavia è comunque possibile sfruttare le caratteristiche tipiche della PCM per rivedere ed ottimizzare le architetture interna dell'array di memoria in termini energetici e di performance. Ad esempio, le letture in PCM non sono distruttive e dunque non vi è bisogno alcuno di riscrivere il *row buffer* appena scaricato, a differenza di quanto avviene nelle DRAM.

Un'altra differenza con una DRAM classica risiede nel fatto che l'architettura PCM può disaccoppiare i circuiti di *sensing* e *buffering*, consentendo una maggiore flessibilità nell'organizzazione dei *row buffers*, i quali possono ospitare linee multiple. Conseguenza non meno importante di questo disaccoppiamento è la possibilità di utilizzare *sense amplifiers* multiplexati: tale multiplexing consente di avere *buffer* più piccoli rispetto alla dimensione dell'array, il quale è definito dal numero totale di *bitlines*. La larghezza del *buffer* è infatti un parametro critico di progettazione: essa determina il numero dei *sense amplifiers*, i quali sono gli elementi che consumano maggiormente in corrente ed in area. L'area altresì salvata può essere riutilizzata per la costruzione di linee aggiuntive di *row buffer*.

La ridotta dimensione dei *buffer* consente inoltre di risparmiare energia in quanto ogni scrittura memorizza più di un bit, eseguita quindi a granularità inferiore.

PCM come Storage System

Al giorno d'oggi le memorie Flash in tecnologia NAND sono vastamente utilizzate nei *Solid-State Disks* (SSD). A differenze dei classici *Hard Disk Drive* (HDD), i quali si basano su dispositivi elettromeccanici, un SSD non ha parti in movimento. Pertanto un dispositivo di questo genere assorbe molta meno potenza rispetto ad un HDD, non emette fastidiosi rumori ed è tollerante a shock e vibrazioni, i quali non costituiscono un problema per il corretto funzionamento del dispositivo. Gli SSD usano inoltre interfacce di memorizzazione standard, quali SCSI e SATA.

Le memorie Flash tuttavia presentano diverse idiosincrasie. Ad esempio, ricordiamo che tali dispositivi non riescono ad eseguire scritture efficienti *in-place*, rompendo la tradizionale astrazione a blocchi che i dispositivi magnetici rotanti fornivano ai livelli soprastanti nella gerarchia di memoria. A complicare il quadro generale si aggiungono le asimmetrie nelle latenze di lettura e scrittura. Poiché uno dei requisiti principali richiesti ai progettisti di memorie Flash è stato quello di renderle compatibili con le interfacce di comunicazione commerciali sopra citate, è stato introdotto il *Flash Translation Layer* (FTL).

Nell'ambito delle memorie non volatili la tecnologia PCM è classificata come *Storage Class Memory* (SCM): tali memorie, a differenza delle Flash hanno due principali caratteristiche:

- sono indirizzabili al byte;
- supportano la scrittura *in-place*

L'uso di una memoria SCM in luogo di una memoria Flash offre dunque diversi vantaggi. Prima di tutto l'abilità di effettuare scritture *in-place* efficienti può semplificare enormemente la progettazione del FTL. La più lunga durata in termini di numero di scritture riduce il bisogno di progettazione ed implementazione di costose politiche di scrittura volte a ridurre al minimo il numero di scritture nel sistema. Infine la ridotta latenza d'accesso porta il non meno considerevole miglioramento in termini di velocità e reattività del sistema.

Le SCM forniranno dunque una sempre maggiore affidabilità e durata; tuttavia il passaggio tra SSD ed SCM renderà ancor più ostico il lavoro di retro compatibilità con le interfacce attuali e i sistemi operativi odierni.

Presente e futuro delle PCM

In conclusione i dispositivi PCM sono una nuova ed emergente tecnologia per quanto riguarda le memorie: attualmente molto utilizzate nei sistemi integrati come rimpiazzo per le più lente memorie FLASH, iniziano ad avere dei forti interessi da parte delle industrie anche per quanto riguarda il loro utilizzo come memorie principali in ambienti diversi da quello *embedded*.

Per quanto tuttavia tali memorie risultino di gran lunga più rapide in termini di tempi di risposta rispetto sia alle memorie flash, sia ai dischi a stato solido e sia soprattutto ai dischi magnetici, le stesse presentano tuttora un percorso di sviluppo lungo e difficoltoso poiché ancora poco utilizzate e quindi relativamente poco studiate e comunque più lente rispetto ad una memoria basata su tecnologia DRAM anche planare e *off chip*.

Questo studio tuttavia considera molto le PCM come parte integrante dei propri sistemi scommettendo dunque in maniera forse azzardata sul futuro sviluppo di questa tecnologia (18); onde inoltre tutelarsi da un eventuale non previsto sottosviluppo si è cercato di affiancare alla “lenta” PCM un dispositivo di archiviazione temporaneo più veloce, la memoria DRAM con approccio tridimensionale.

3D-DRAM come Last Level Cache

Nei sistemi attuali, oltre alle cache di primo e secondo livello generalmente private per ogni core comprendente il processore fisico, esiste una cache di terzo livello, riferita nel seguito con L3 o LLC, molto più grande delle precedenti e condivisa tra tutti i core. Le dimensioni di tale cache sono generalmente nell’ordine di $4 - 12$MB e sono affiancate dalla memoria RAM principale.

Nei sistemi in studio in questo documento tuttavia ciò che affiancherebbe la L3 risulta essere una memoria principale basata su PCM: è ovvio quindi, che confrontando dimensioni piccole, nell’ordine della decina di megabyte, con le enormi dimensioni di

una PCM e, soprattutto, considerando il *gap* di velocità tra le tecnologie, che la sola PCM non è sufficiente a garantire alte prestazioni al sistema.

Si è pertanto scelto di utilizzare come LLC non una memoria SRAM come nei sistemi tradizionali bensì una 3D-DRAM *stacked*.

L'approccio è facilmente riconducibile a quello utilizzato per l'architettura CMM e, in effetti, molte sono le similitudini tra le due tipologie di memoria; l'approccio 3DLLC tuttavia disaccoppia la bivalenza tra cache e memoria principale adottata dall'architettura CMM e mantiene distinte le due entità.

Coerenza della 3D-DRAM-LLC

La last level cache è una memoria di sua natura condivisa. Ciò significa che, ad un certo momento nella vita operativa della memoria, più processi e quindi più processori e/o più core potrebbero voler accedere allo stesso dato.

Si immagini per semplicità un sistema composto da un processore fisico nel quale sono presenti 2 core. Ognuno dei core possiede la sua *Instruction-Cache* privata di primo livello, la sua *Data-Cache* privata di primo livello e un'altresì privata cache di secondo livello.

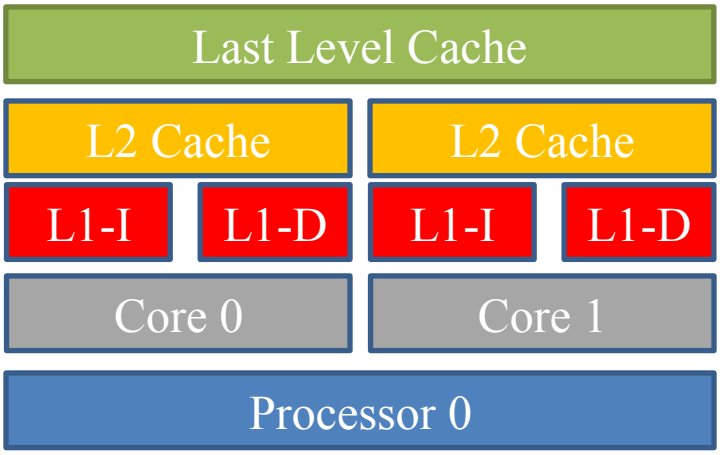

Figura : Architettura semplificata con LLC

Si supponga ora che avvengano le seguenti operazioni in sequenza:

1. Il *core* 0 richiede un dato ad un certo indirizzo fisico di memoria condivisa
    a. Il dato non risulta essere in cache di livello 1

   b. Il dato non risulta essere in cache di livello 2
   c. Il dato non risulta essere in cache di livello 3
2. Il dato viene prelevato dalla memoria principale e viene portato in tutti e tre i livelli di cache
3. Il processore modifica il dato cambiandone il valore nella sua L1
   a. La politica write back impedisce al dato di essere riscritto nella L2 del *core* 0 o in L3
4. Il core 1 richiede lo stesso dato relativo al precedente indirizzo fisico di memoria condivisa
   a. Il dato non risulta essere in cache di livello 1
   b. Il dato non risulta essere in cache di livello 2
   c. Il dato risulta essere in cache di livello 3 e viene prelevato dal *core* 1 e posto nei suoi livelli privati di cache.

A questo punto è necessario fermarsi e chiedersi se effettivamente queste operazioni sono coerenti tra loro. La risposta è negativa in quanto al punto 4c il dato che il *core* 1 vede nelle proprie cache private è il dato "vecchio" non modificato dal *core* 0.

Per risolvere questo problema il sistema oggetto di studio utilizza il protocollo MESI (19) per mantenere la coerenza tra i vari livelli di cache: tramite *snooping* del bus condiviso che porta i core alle loro cache sarà quindi la cache di primo livello del *core* 0 a rispondere alla corrispettiva cache dello stesso livello del *core* 1 notificandole che è ella stessa ad avere la più aggiornata copia del dato.

Successivamente, nel momento in cui sarà il core 1 a modificare il dato, la cache di primo livello (o comunque la cache di più basso livello che contiene il dato) notificherà a tutte le cache dello stesso livello di invalidare il dato appena modificato. Si noti come questa operazione veniva comunque eseguita anche al punto 3 quando il *core* 0 ha modificato il dato per primo: questa operazione tuttavia non aveva avuto alcun effetto a causa della mancanza del dato nella cache di primo livello del *core* 1.

Informazioni più dettagliate sul protocollo MESI possono essere trovate in (19) o in (20).

Componentistica della 3D-LLC

La memoria 3D come cache di ultimo livello è di per sé un componente unico; ciò nonostante però vi è da considerare la discordanza che si ha tra il termine cache, generalmente associato ad un tipo di tecnologia basato su logica statica, e il termine DRAM che è notoriamente un tipo di tecnologia basato invece su logica dinamica. La 3D-LLC quindi può essere idealmente e praticamente suddivisa in due componenti distinte, la porzione costruita mediante logica statica che realizza il meccanismo di indicizzazione ad insiemi tipico di una cache e la porzione che invece contiene i dati veri e propri e che realizza la memoria in sé.

SRAM

Lo scopo principale della SRAM all'interno del componente 3D-LLC è quello di realizzare l'indicizzazione *cache-like* della memoria: a tale fine è necessario definire, all'interno dell'indirizzo, fisico o virtuale che sia, una parte designata ad essere TAG.

Inoltre, rimpiazzando la classica cache di terzo livello con questa basata su tecnologia DRAM, si nota come con questa sostituzione, si risparmi, in termini di area occupata, quasi il 50% dell'area totale processore (cfr. Figura).

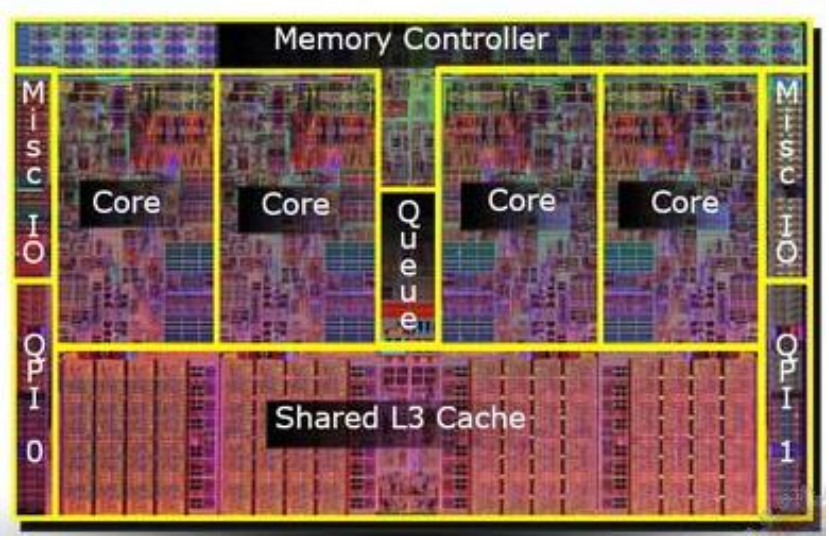

Figura : Architettura di un processore i7 Nehalem (curtesy of Intel)

Al posto della cache di livello 3 che doveva precedentemente contenere sia la struttura dei *tag* sia i dati è possibile inserire la parte SRAM del sistema in

considerazione e, grazie alla grande area a disposizione, è possibile indirizzare una quantità di memoria molto maggiore.

La generica linea di cache appare come segue:

Tabella : Struttura di una linea di cache nella SRAM della 3DLLC

Physical Tag **≤ 49 bit**

Si noti come la LLC è stata pensata in maniera tale da essere indirizzata e taggata fisicamente e come non sia effettivamente memorizzata alcuna informazione riguardo l'indirizzo in 3D-DRAM. La prima scelta, in concordanza con le politiche attuali di progettazione, consente innanzitutto l'accesso a tale memoria solo nel momento in cui si ha a disposizione l'indirizzo fisico del dato richiesto. Tale indirizzo dovrà preventivamente essere ricavato tramite un TLB o una struttura Hardware/Software equivalente. Una volta indirizzato il *set* contenente tante entrate quanta è l'associatività a insiemi della cache sarà possibile utilizzare un'ulteriore parte dell'indirizzo fisico, ossia il *Physical Tag* per localizzare la giusta entrata nella SRAM della cache.

Se il dato è presente in cache allora ciò che accade è che la DRAM posta sopra il processore verrà attivata tramite l'indirizzo calcolato a partire dalla linea di cache che ha scatenato la hit. Questo procedimento, denominato *index trick* è stato già trattato nel paragrafo relativo alla SRAM associata alla CMM e pertanto non verrà riproposto.

Se invece il dato non è presente in cache allora verrà scatenata una *miss* e si procederà al normale accesso in memoria.

DRAM

Il vero e proprio contenitore dei dati per quanto riguarda la LLC è una 3D-DRAM. La tecnologia per la realizzazione di tale memoria è ovviamente quella *True-3D* per i motivi ampiamente esposti nell'introduzione di questo documento.

Lo schema concettuale definitivo per l'accesso a questo tipo di memoria risulta quello riportato nella figura seguente.

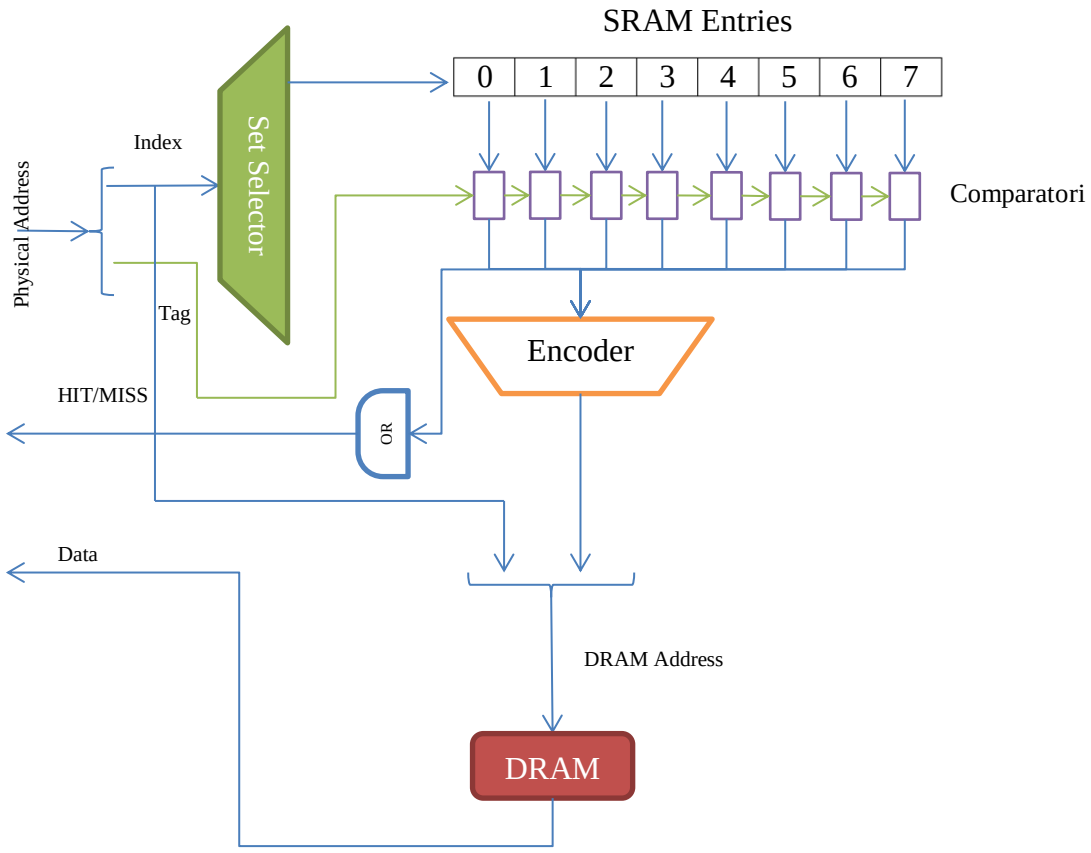

Figura : Struttura del sistema 3D-DRAM come LLC

La grandezza di una singola linea di memoria è fissata, per la LLC, a 1024 byte ossia 8 volte la grandezza di una linea di cache di livello inferiore. La memoria principale da collegare a questo sistema dovrà quindi essere configurata in maniera tale da avere come grandezza minima di trasferimento da e verso la L3 pari proprio a questo valore.

Row Buffer

Anche in questa DRAM, sebbene usata come cache, saranno ovviamente presenti i *row buffers*, creando effettivamente un livello intermedio di cache. La strutturazione dei *row buffers* è esattamente uguale a quella utilizzata nella CMM e pertanto non verrà riproposta anche in questo paragrafo.

Componentistica della PCM-MM

Come già accennato precedentemente si è scelto di utilizzare come memoria principale, nell'ambito della configurazione che utilizza tra l'altro la 3D-DRAM come LLC, una memoria basata sulla tecnologia a cambiamento di fase (PCM-MM).

I primi prodotti di questo tipo realizzati e commercializzati da Micron (21) sono internamente strutturati e quindi esternamente visibili come delle classiche memorie RAM (12). Questo permette agli attuali controllori della memoria montati sui processori odierni di non richiedere praticamente alcun intervento di riprogettazione per adattarsi al nuovo componente che risulta quindi, in tutto e per tutto, *Plug and Play*.

Lo studio realizzato su tale componente pertanto si basa su questa caratteristica interna delle PCM ed è stato organizzato in maniera da mantenere l'accesso alla memoria principale identico a come era stato effettuato nell'ambito della CMM: la PCM dunque sarà dotata di una SRAM che ne realizza l'indirizzamento *cache-like* e di un TLB che avrà il compito di tradurre l'indirizzo virtuale in arrivo dal processore in un indirizzo fisico relativo alla PCM stessa. In quest'ottica dunque restano validi tutti i discorsi fatti precedentemente sulla CMM e ovviamente restano valide tutte le considerazioni fatti sulle eventuali scelte progettuali effettuate.

Dimensionamento delle sottopagine

Particolare precisione progettuale richiede il dimensionamento di una sottopagina: è stato detto che la dimensione di una linea di memoria della last level cache è stata fissata a 1024 byte. La CMM che assiste la PCM tuttavia ha una dimensione di sottopagina che è di soli 128 byte. Questo significa che ogni qual volta vi è una richiesta di un dato dalla memoria principale verso la *last level cache* compito della memoria sarà quello di trasferire in *burst* non una, bensì otto sottopagine consecutive.

Si sarebbe potuto in effetti implementare la *last level cache* in modo che essa potesse memorizzare linee di memoria da 128 byte così come le cache di più basso livello, tuttavia questo avrebbe richiesto, causa la grande dimensione di tale cache, una SRAM di indicizzazione estremamente grande. Infatti, supponendo sottopagine da 128 byte e una totale dimensione della cache di 8 GB (la più grande dimensione considerata

nelle simulazioni) si avrebbe una SRAM, da realizzare tramite più costosa tecnologia CMOS, la cui grandezza è data da:

$$\#SRAM_{entries} = \frac{DRAM_{size}}{SUBPAGE_{size}} = \frac{8 \cdot 2^{30} bytes}{128 \frac{bytes}{entry}} = 67108864\ entries\ = 64Mentries$$

e dunque una dimensione massima di:

$$SRAM_{size} = \#SRAM_{entries} \cdot ENTRY_{size} = 67108864\ entries\ \cdot \left\lceil \frac{49\ bit}{8 \frac{bit}{byte}} \right\rceil = 448Mbyte$$

Utilizzando invece *cache line* da 1024 byte si ha:

$$\#SRAM_{entries} = \frac{DRAM_{size}}{SUBPAGE_{size}} = \frac{8 \cdot 2^{30} bytes}{1024 \frac{bytes}{entry}} = 8388608\ entries\ = 8Mentries$$

e una dimensione massima di:

$$SRAM_{size} = \#SRAM_{entries} \cdot ENTRY_{size} = 8388608\ entries\ \cdot \left\lceil \frac{49\ bit}{8 \frac{bit}{byte}} \right\rceil = 56Mbyte$$

che è 8 volte inferiore a quella che si avrebbe con *cache line* della dimensione di una sottopagina. (Il calcolo è immediato se si pensa che $\frac{1024}{128} = 8$)

I Row Buffers nella PCM-MM

Come si è già menzionato in precedenza la PCM-MM è internamente strutturata come una normale ed attuale memoria RAM. Questo ovviamente significa che sono presenti anche i *row buffers* che creano un ulteriore livello di *caching* di fatto portando l'intera architettura di memoria 3D-DRAM-LLC + PCM-MM da due a quattro livelli:

- Row Buffer 3D-DRAM
- 3D-DRAM
- Row Buffer PCM
- PCM

Un'importante considerazione da tenere conto nei *row buffers* della PCM è che quest'ultima, essendo una memoria permanente, non necessita di *refresh* né tantomeno è

caratterizzata da operazioni di lettura distruttive: ciò significa che una volta letto e trasportato nel *row buffer*, il dato resterà lì indefinitamente sino allo spegnimento del sistema o finché un ulteriore dato dallo stesso banco verrà richiesto.

Miglioria aggiuntiva proposta da (12) è stata quella di disaccoppiare i *sense amplifiers* dai *row buffers* veri e propri in maniera tale da poter realizzare i medesimi multiplexando i vari dati provenienti dai diversi *sense amplifiers* che fanno capo a quel buffer. Questa operazione è possibile tuttavia solo nei sistemi PCM e non in quelli DRAM in quanto nelle memorie classiche l'operazione di lettura distruttiva richiede, pena la perdita delle unità informative, lo scaricamento di un'intera linea di memoria e la sua successiva riscrittura nelle celle pertinenti.

Visione di insieme

Una visione a blocchi dell'intero sistema 3D-DRAM-LLC + PCM-MM è visibile nella figura seguente (22)

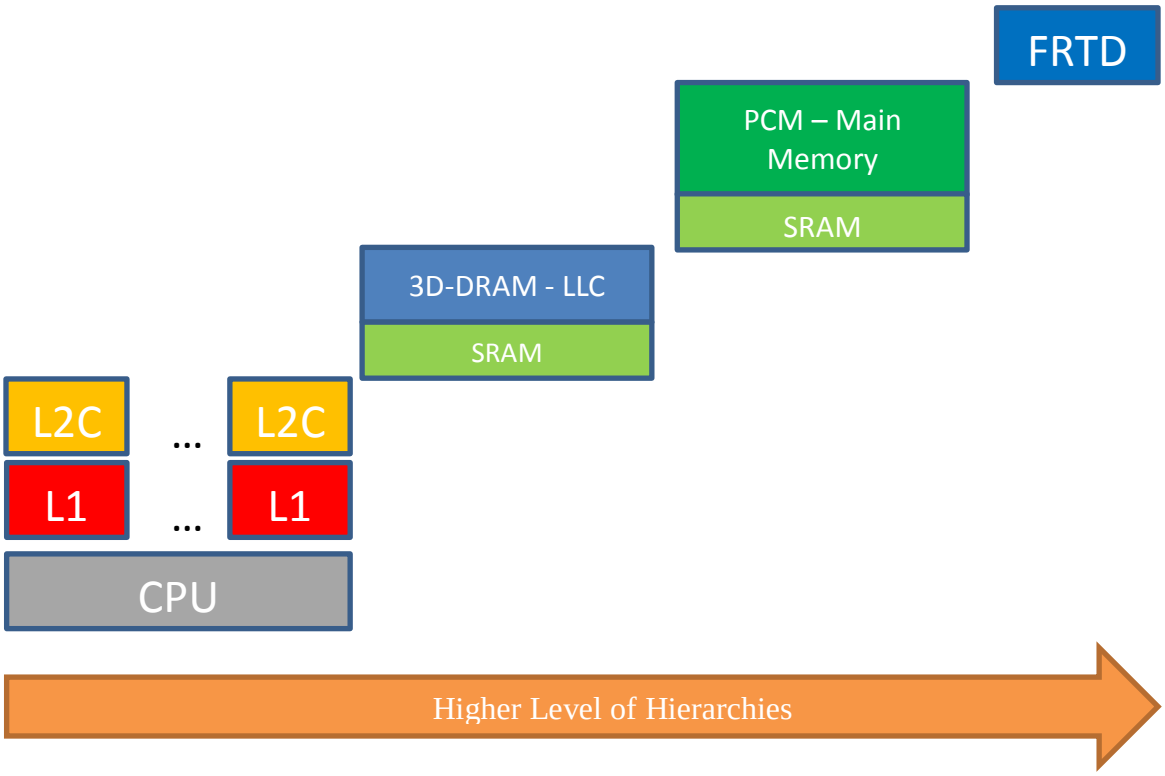

Figura : Visione di insieme del sistema 3D-DRAM-LLC + PCM-MM

Dalla figura si nota come anche in questo caso esista un dispositivo di memorizzazione di massa dei dati atto a contenere le pagine non residenti in memoria

principale ossia in PCM. Rimanendo nell'ambito della CMM si è scelto di continuare ad utilizzare un tipo di memoria FRTD per il *back storage* ed utilizzare quest'ultimo come dispositivo per lo swap. Ulteriori informazioni sulle FRTD sono contenute nel sotto paragrafo ad esse relativo riportato nella sezione della CMM.

Phase Change Memory e 3D-DRAM as Part of Main Memory

Nella già esposta architettura della CMM si è finora preso in considerazione l'idea di utilizzare come memoria principale o la 3D DRAM o la PCM. Nulla vieta tuttavia di cercare di utilizzare entrambe le soluzioni cercando di trarre il meglio da ognuna di esse.

In particolar modo si è cercato di esplorare una soluzione che potesse unire la velocità della 3D DRAM ai bassi consumi e alla scalabilità della PCM. Si è dunque pensato ad un dispositivo ibrido che possa fornire l'astrazione di una CMM classica in cui però la velocità di risposta del dispositivo deputato a memoria possa presentare pagine fisiche potenzialmente contigue in dispositivi con latenze e caratteristiche diverse.

Il primo e nonché più ovvio problema riguarda l'organizzazione fisica di una tale memoria, prescindendo dall'architettura della CMM: il sistema CPU – Memorie Cache deve essere a conoscenza di tale organizzazione, oppure è meglio presentare l'astrazione di una memoria unificata e gestire staticamente in hardware tale fusione? La risposta a tale domanda non è banale e richiede una serie di considerazioni di carattere architetturale e prestazionale.

Rendere il sistema operativo e dunque il sistema CPU – Memorie Cache conscio di tale distinzione e fornire dunque la possibilità di gestire via software una tale gerarchia di memoria, comporta da un lato una gestione dinamica e riconfigurabile delle pagine virtuali, consentendo ad esempio al sistema operativo di rilocare le pagine virtuali più frequentemente accedute nella memoria più veloce (3D DRAM), ma implica dall'altro lato una drastica riorganizzazione del sistema operativo stesso, la quale può essere costosa e non sufficientemente adeguata nello sfruttare le possibilità che tale organizzazione di memoria offre.

Se si vuole rendere il sistema operativo completamente inconscio della presenza di questa divisione si dovrà aggiungere della logica ausiliaria al classico controllore di memoria in modo tale che possa gestire le operazioni di memoria traducendo gli indirizzi

fisici di memoria (che il sistema operativo ha generato considerando la memoria fisica come un unico grande blocco) negli indirizzi fisici relativi al dispositivo di memoria corretto. Durante questa trattazione chiameremo questo nuovo indirizzo fisico indirizzo fisico-ibrido di memoria. In Figura è presentato un semplice schema che riassume quanto appena detto.

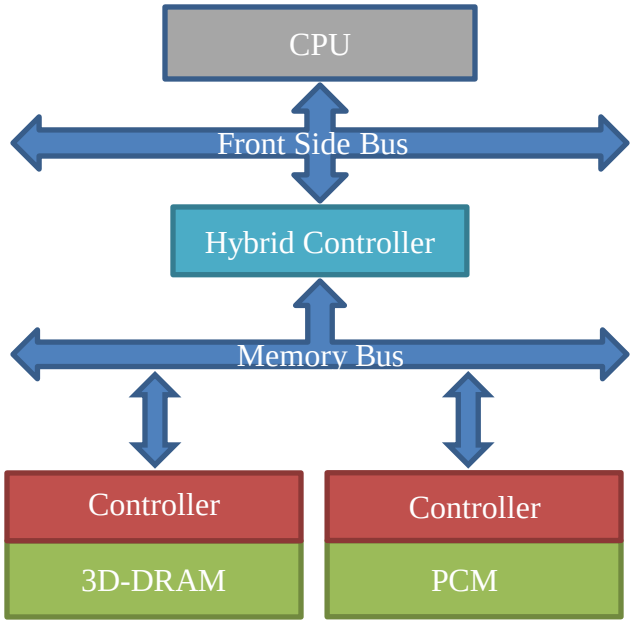

Figura : Architettura di memoria ibrida

Rendere questo controllore ibrido di memoria riprogrammabile permetterà l'utilizzo di differenti politiche di smistamento degli indirizzi nelle due memorie.

Politiche di smistamento hardware

La politica di smistamento più semplice potrebbe essere quella in cui le operazioni di memoria interessino in modo completamente casuale sia la parte di memoria in 3D DRAM sia la parte di memoria in PCM. Nell'utilizzare questo tipo di politica si dovrà ovviamente tenere traccia in quale delle due memorie siano memorizzate i dati e questo introdurrebbe un *overhead* non trascurabile e non porterebbe un significativo incremento di performance. In Figura è riportato un semplice schema che mostra l'organizzazione delle pagine di memoria e il loro smistamento casuale.

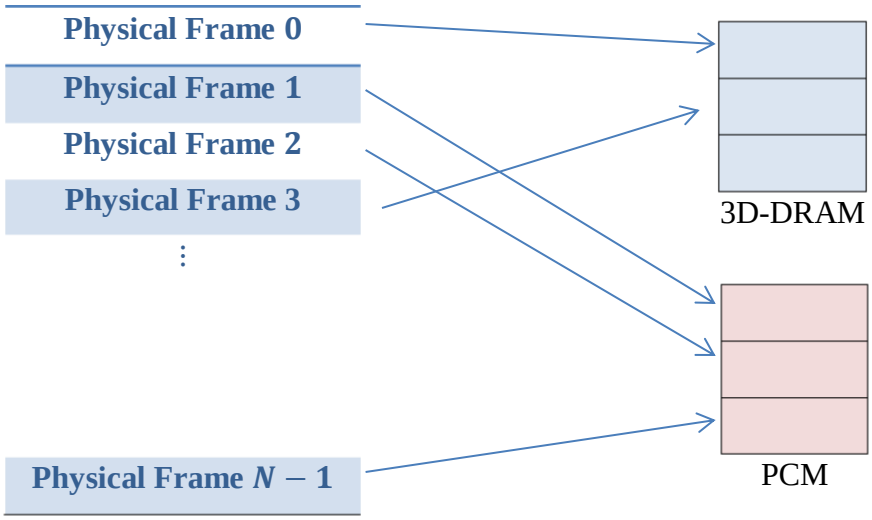

Figura : Astrazione della memoria fisica

Politica di smistamento a gruppi

Una strategia più efficiente potrebbe essere quella in cui le pagine risultino organizzate in sottoinsiemi, ai quali ci si riferirà col nome di gruppi, all'interno della memoria unificata nei quali saranno presenti un numero prefissato di pagine relative alle due memorie principali. Il numero di pagine di ciascun dispositivo di memoria presente in ogni gruppo sarà dettato dal rapporto delle grandezze tra le due memorie. Si supponga ad esempio che il rapporto tra la dimensione della 3D DRAM e la PCM sia:

$$\frac{3dDRAM_{size_{GB}}}{PCM_{size_{GB}}} = \frac{a}{b}$$

Questo comporterà che all'interno di ogni gruppo saranno presenti a pagine di 3D DRAM e b pagine di PCM, per un totale di :

$$\#groups = \frac{3dDRAM_{size_{GB}} + PCM\_size_{GB}}{(a+b) \cdot PAGE_{size}}$$

A questa politica è stato dato il nome di politica di smistamento a gruppi. Verrà analizzato nel paragrafo relativo alle simulazione come cambiare il rapporto tra i due tipi di memoria, influirà non poco sulle prestazioni del sistema e sul consumo energetico dello stesso. Per comprendere come l'indirizzo fisico della memoria totale possa essere tradotto nel relativo indirizzo ibrido di memoria, ci si riferisca alla figura seguente.

Tabella : Indirizzo fisico proveniente dal processore

Page Number	Page Offset

Si immagini una memoria composta da 1 GB di 3D DRAM e 4 GB di PCM: in tal caso il rapporto tra le due memorie sarà $\frac{1}{4}$ si otterranno quindi gruppi di 5 pagine fisiche in cui la prima risiederà nella 3D DRAM e le rimanenti quattro nella memoria PCM.

Dividendo a questo punto il campo *PageNumber* in figura per il numero di pagine in un gruppo, si ottiene l'indice del gruppo di pagine all'interno della memoria unificata che denominato *groupNumber*; effettuando poi un'operazione di modulo tra *PageNumber* e il numero di pagine in un gruppo si ottiene il numero di pagina all'interno del gruppo stesso che denominato *groupOffset*. Nel caso in cui il valore di *groupOffset* fosse minore del numero di pagine di 3D DRAM presenti in ciascun gruppo allora il dato ricadrà in 3D DRAM e il suo indirizzo fisico ibrido di memoria sarà:

$$3dDRAM_{address} = groupNumber * PAGE_{size} + groupOffset$$

altrimenti il dato ricadrà in PCM ed analogamente il nuovo indirizzo fisico ibrido di memoria sarà:

$$PCM_{address} = groupNumber * PAGE_{size} + groupOffset - \#3dDRAM_{pages_{group}}$$

Il meccanismo finora indicato viene riportato schematicamente nella figura seguente al fine di fornirne una più immediata e semplice comprensione al lettore.

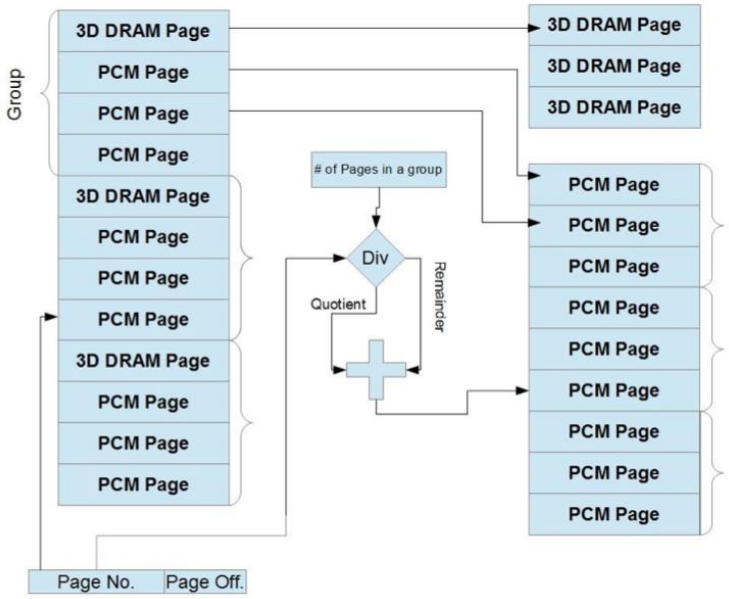

Figura : Politica di smistamento a gruppi

L'implementazione hardware di un tale meccanismo potrebbe risultare dispendiosa, quindi si cercherà nei capitoli successivi di trovare una politica più efficiente in termini di operazioni hardware.

Group Mixed Policy Alternativa

La politica studiata in questo capitolo, pur ponendo delle forti restrizioni sulle dimensioni delle due memorie, si avvale di più semplici ed efficienti operazioni hardware per traduzione dell'indirizzo fisico nel relativo indirizzo fisico ibrido di memoria. Semplicemente considerando alcuni bit immediatamente superiore al *PageOffset* dell'indirizzo fisico per discernere il posizionamento della pagina fisica in una delle due memorie. Si consideri ad esempio una quantità totale di memoria principale di α GB, con la limitazione che $\alpha = 2^n$ con $n \in \mathbb{N}$.

Dovremo allora considerare b bit subito superiore al *PageOffset*: si avrà quindi un numero totale di pagine all'interno di un singolo gruppo pari a 2^b. Allora ogni combinazione di b bits potrà essere utilizzata dal controllore ibrido di memoria per localizzare la pagina indirizzata nel corretto dispositivo di memoria.

Il relativo indirizzo fisico ibrido di memoria all'interno di uno dei due dispositivi di memoria potrà essere calcolato considerando $c < b$ bit subito superiori al *PageOffset*. La Figura mostra questa politica alternativa. In questo tipo di politica di smistamento il controllore di memoria ibrido potrà contenere al suo interno una memoria ROM per il *mapping* tra le combinazioni possibili di b bits e il dispositivo di memoria dove si vuole posizionare la pagina. Questo tipo di politica che prevede comunque la divisione dell'intero spazio di memoria fisica in gruppi ma con un diverso smistamento delle pagine all'interno di ogni singolo gruppo è stata nominata politica di smistamento a gruppi alternativo. Questo algoritmo non è stato implementato nelle simulazioni per le forti restrizioni sulla dimensione dei singoli dispositivi di memoria necessarie.

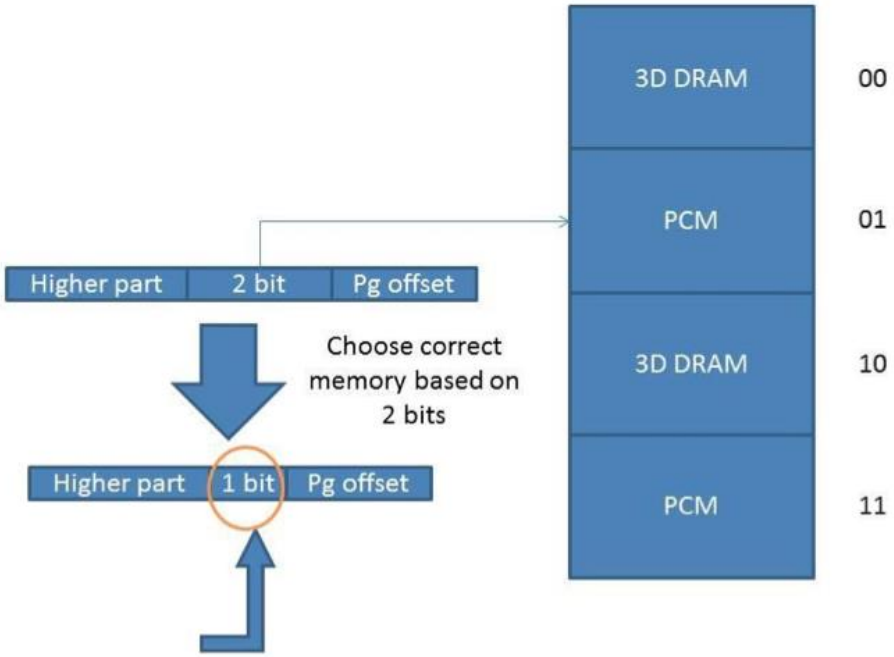

Figura : Politica a smistamento alternativa

Politica di Smistamento Forte

Un'altra politica di smistamento considerata in questo studio prevede un'alternanza delle pagine di 3D DRAM e PCM considerando questa volta un unico grande gruppo, grande come la dimensione dell'intera memoria principale (somma delle due singole memorie). Questa politica è stata nominata politica di smistamento forte. Le pagine di 3D DRAM e PCM alternate una dopo l'altra finché una delle due, la più piccola ovviamente, non termina, dopodiché non ci sarà più alternanza e saranno presenti

solamente le pagine relative al dispositivo di memoria più grande. Un esempio chiarificatore di questo approccio è presentato di seguito.

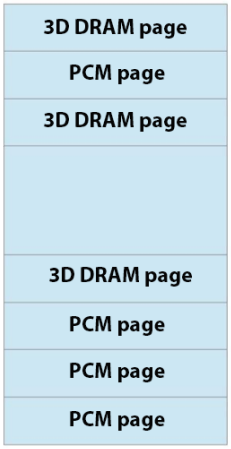

Figura : Politica di Smistamento Forte

Questo tipo di politica risulta implementabile con semplici operazioni hardware, poiché basterà controllare la parità del campo *PageNumber* dell'indirizzo fisico di memoria per discriminare se la pagina debba riferirsi al dispositivo di memoria 3D o alla PCM. Ovviamente se il campo *PageNumber* supera in grandezza il numero di pagine che è possibile memorizzare in 3D DRAM, sia le pagine dispari che le pagine pari si riferiranno alla memoria PCM.

Di seguito viene presentato un breve algoritmo in pseudo-codice per il calcolo dell'indirizzo fisico ibrido di memoria a partire dall'indirizzo fisico (si consideri un indirizzo fisico di arbitraria lunghezza con un *PageOffset* di 15 bit)

```
// check the parity of the 16th bit
if((PhysicalAddress & (1<<16)) == 0)
{
        // 16th bit is 0
        // PageNumber is even so it belongs to the 3D DRAM
        //Hybrid Address = Physical Address without the 16th bit
}
else
{
        // 16th bit is 1
        // PageNumber is even so it belongs to the PCM
        //Hybrid Address = Physical Address - #pages(3D_DRAM)
}
```

Politica di smistamento contiguo

Un'altra politica di smistamento studiata è quella denominata politica di smistamento contiguo. Si può infatti considerare una struttura secondo la quale la prima parte della memoria fisica totale è implementata dalla memoria 3D DRAM e la restante dalla memoria PCM. In tal caso è possibile rendere conscio il sistema operativo di tale distinzione lasciandogli l'incarico di poter scegliere a proprio piacimento il posizionamento delle pagine virtuali in memoria fisica.

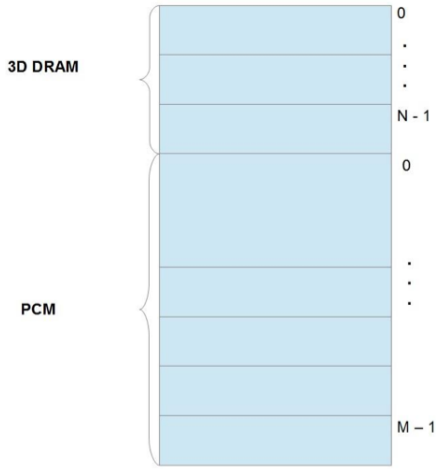

Figura : Politica di smistamento contiguo

Politica di smistamento basata sugli accessi alle pagine

La già accennata politica di smistamento basata sul numero di accessi ad una pagina è quella che intuitivamente sembra essere la più efficiente. Essendosi questo studio proposto l'obbiettivo di rendere trasparente al sistema operativo l'architettura ibrida della memoria, si rende dunque necessaria una qualche implementazione hardware del meccanismo proposto. Si consideri uno scenario tipico dell'ambiente CMM in cui si hanno pagine da 32KB, una 3D DRAM da 2GB e una PCM da 8GB: si ha dunque una memoria totale da 10GB nella quale risiedono 327680 pagine fisiche. A questo punto ciò che si vorrebbe la memoria facesse è che questa tenga traccia degli accessi a ciascuna della pagine fisiche e che portasse dinamicamente le più accedute nella memoria più veloce effettuando quindi uno scambio di traduzione al volo. Per ogni pagina bisognerebbe dunque memorizzare un contatore di quantomeno un byte e l'indirizzo in una delle due memorie in cui la pagina fisica è stata realmente memorizzata, per un totale

di $1 + 6 = 7$ byte a pagina. Questo comporta l'utilizzo di una struttura hardware che memorizza $7 \cdot 327680 = 2293760$ byte, i quali corrispondono a $\sim$2 MB: bisogna dunque tenere conto dell'*overhead* di area e potenza che una tale struttura può introdurre.

Da non sottovalutare è anche l'*overhead* che introduce lo spostamento delle pagine meno utilizzate da 3D DRAM a PCM e lo spostamento delle pagine più utilizzate da PCM a 3D DRAM, nonché i tempi di ricerca e la complessità dell'algoritmo di scelta dello scambio delle pagine. Da sconsigliare in ogni caso è lo spostamento da 3D DRAM a PCM per poter preservare quanto più possibile la durata di quest'ultima, la quale come ben noto ha a disposizione un numero limitato di scritture per cella.

Una possibile soluzione a questo *overhead* è quella dare la priorità minima agli spostamenti tra le pagine, i quali possono avvenire solamente durante i cicli liberi del bus.

Politiche di smistamento software

Nel caso in cui si volesse rendere il sistema operativo conscio della divisione della memoria principale in due sottosistemi di memoria distinti e con *delay* e consumi di potenza nettamente differenti, si aprirebbe il campo ad un ampio insieme di politiche implementabili a livello di nucleo di sistema. Chiaramente questo tipo di approccio comporterebbe una netta modifica del sistema operativo.

Si potrebbe pensare di fornire agli sviluppatori di applicazioni un sottoinsieme di API per poter specificare dove voler salvare il proprio spazio di memoria o altresì pensare di poter permettere agli utenti di selezionare una modalità di avvio dei processi fast, in cui tutto lo spazio di memoria del processo sarà salvato nella più veloce 3D DRAM. In questo tipo di politica lo smistamento potrebbe semplicemente essere fatto sul campo ASID degli indirizzi di memoria. Come detto sopra, questo tipo di approccio apre il campo ad un largo insieme di politiche, che andranno opportunamente testate e valutate anche sotto l'importantissimo punto di vista della protezione.

STRUMENTAZIONE UTILIZZATA

Cacti

CACTI (23) è un potente strumento di modellazione di memorie sviluppato dalla HPLabs, sezione ricerca e sviluppo della nota società HP. CACTI genera un modello che integra al suo interno tutta una serie di informazioni utili alla corretta valutazione della architettura di memoria sotto studio, quali:

- *memory access time*
- *cycle time*
- *area*
- *leakage power*
- *dynamic power*

attraverso le quali il progettista può far fronte agli svariati *trade-off* derivanti dalle scelte effettuate sui parametri di progettazione. Man mano che i processi produttivi si andavano evolvendo, CACTI ha subito pesanti modifiche per adeguarsi agli standard produttivi più recenti, cambiamenti che lo hanno reso un ottimo strumento per lo studio di nuove architetture di memoria o per la modellazione delle attuali tecnologie in termini di memorie SRAM e memorie DRAM.

Molti progettisti, per far fronte alle proprie esigenze di ricerca di innovative architetture di memoria, hanno modificato il codice sorgente di CACTI (codice `C++`) per poter creare degli ambienti di simulazione per il test di memorie di ultima generazione quali le 3D *Stacked* DRAM o le *Phase Change Memories*. CACTI mette a disposizione inoltre una interfaccia web le effettuare le proprie simulazioni, con un livello di dettaglio nella configurazione della propria architettura nettamente inferiore rispetto a quello che si ha a disposizione utilizzando il pacchetto contenente i sorgenti.

Parametri di Configurazione

Si prenderà adesso in considerazione un file di configurazione di CACTI 6.5 generico per evidenziare tutti i parametri disponibili per il progettista, parametri che variano a seconda del tipo di architettura di memoria da simulare; in particolare la versione che verrà trattata in questo capitolo permette la simulazione di 3 tipi di memoria:

1. **Cache**, comprensiva quindi di un *tag array*, selezionabile tramite la parola chiave *cam*
2. **Ram**, modello simile ad un *Register File*, selezionabile tramite la parola chiave *ram*
3. **Main memory**, memoria senza *tag array* cui ogni accesso avviene alla granularità di una pagina di memoria, selezionabile tramite le parole chiave *main memory*

Nel seguito si farà riferimento solo alla prima e alla terza architettura.

Nei paragrafi successivi verranno esaminati alcuni del principali parametri utilizzati per la modellazione dell'architettura di memoria desiderata, Main e Cache Memories, all'interno di CACTI.

Parametri di configurazione comuni a Main Memories e a Cache

`size <memsize>`

- Per la modellazione di memorie cache questo parametro specifica la dimensione in bytes della sola 'parte dati' della memoria cache.
- Per la modellazione di memorie principali questo parametro specifica la dimensione in GB della memoria.

`block size <blockSize>`

- Per la modellazione di memorie cache questo parametro specifica la dimensione in bytes della parte dati di una singola linea di cache
- Per la modellazione di memorie principali questo parametro specifica la dimensione in bytes di una singola linea di memoria

`UCA bank count <bankCount>`

- Il parametro indica il numero totale di banchi, per ogni modulo di memoria, connessi attraverso un singolo bus centrale, "*stripe bus*".

`technology (u) <technology>`

- Il parametro indica la dimensione minima di gate di ogni singolo transistor utilizzata nel processo costruttivo delle celle

`Data array cell type <cellType>`

`Data array peripheral type <peripheralType>`

- I parametri possono assumere valori diversi in base al processo produttivo considerato, e si distinguono in relazione al costo di produzione, alle performance ottenibili e al consumo di potenza. In particolare il primo si riferisce al tipo di cella di memoria utilizzata mentre il secondo si riferisce al tipo di logica aggiuntiva in supporto alla cella di memoria (*latches*, *sense amplifiers*, eccetera). Si distinguono tre valori:
    1. **itrs-hp**: *International Technology Roadmap for Semiconductors High Performance* - Processo ad alte prestazione, tipico delle memorie cache
    2. **itrs-lstp**: *International Technology Roadmap for Semiconductors Low Standby (Static) Power* - Processo per medie prestazioni rivolto ad un basso consumo di potenza, tipico della logica di supporto nelle memorie DRAM
    3. **comm-dram**: *Common DRAM* - Processo per medie prestazioni e medio consumo di potenza, tipico di celle di memorie DRAM.

`output/input bus width <busWidth>`

- Il parametro indica la larghezza del bus di input/output per il modulo di memoria

`operating temperature (K) <temperature>`

- Il parametro indica la temperatura in gradi Kelvin prevista di lavoro del modulo di memoria. Verrà utilizzato dal simulatore per determinare la densità delle celle di memoria e dei bus interni.

`cache type <type>`

- Il parametro indica il tipo di memoria che si vuole modellare; può assumere i valori `cache` oppure `2D main memory` per la modellazione degli omonimi tipi di memoria

Altri parametri disponibili sono utili per specificare dei vincoli di prestazione o consumo di potenza da cui il modello si può al massimo discostare, e non verranno trattati in questi capitoli.

Parametri di configurazione per le memorie cache

`associativity <associativity>`

- Il parametro indica l'associativa per i set all'interno della memoria cache. Per ottenere una cache *fully-associative* il parametro deve essere settato a 0.

`Tag array cell type - <cellType>`
`Tag array peripheral type - <peripheralType>`

- Questi parametri sono utili a specificare il dettaglio del processo produttivo per le celle di memorie (e la logica aggiuntiva) che andranno a contenere la parte *tag* per ogni linea di cache. I valori assumibili da questi parametri sono stati già trattati precedentemente e non verranno riproposti in questa sede.

`tag size (b) <tagsize>`

- Il parametro indica la grandezza misurata in bit della parte *tag* relativa ad ogni linea di memoria cache.

`access mode (normal, sequential, fast) - <mode>`

- Il parametro permette la scelta di una politica di accesso alle parti *tag* e *data* delle linee di cache. In particolare:
    - **fast**: l'accesso alla parte *tag* e alla parte *data* avvengono in parallelo
    - **sequential**: la parte *data* viene acceduta solo dopo aver acceduto alla parte *tag*

- **normal**: l'accesso alle parti *tag* e *data* avvengono in parallelo, e il blocco data selezionato viene inviato in *broadcast* nel *bus* H-TREE solo dopo aver ricevuto il segnale dal *tag array* (*hit* o *miss*)

`Optimize ED or ED^2 (ED, ED^2, NONE): <value>`

- Il parametro permette la scelta del tipo di ottimizzazione per il *tag* array:
  - **ED**: Ottimizzazione *Energy-Delay Product*
  - **ED^2**: Ottimizzazione *Energy-Delay Square Product*

`Cache model (NUCA, UCA) - <model>`

- Il parametro permette di scegliere il tipo architettura nella quale si andrà ad inserire la memoria cache. Questo parametro è utile per ottimizzare l'architettura di memoria nel caso in cui si preveda o meno il suo utilizzo in una architettura multiprocessore.

Parametri di configurazione per le memorie principali

`page size (bits) <pageSize>`

- Il parametro permette di scegliere la grandezza della pagina di memoria che si prevede di utilizzare nel sistema dove la architettura di memoria andrà installata. Questo permette di ottimizzare la grandezza della *bitline* all'interno di ogni banco di memoria, ovvero il numero di colonne all'interno di ogni banco.

`system frequency (MHz) <frequency>`

- Il parametro permette la specifica del I/O Bus Clock. Tipici valori per questo parametro possono essere facilmente reperibili sui portali dei maggiori produttori di memorie DRAM.

Output di CACTI

In questo capitolo verranno analizzati i valori di output generati da CACTI, considerando solamente i principali parametri utili alla classificazione di una memoria in termini di velocità di lettura e scrittura e in termini di energia consumata per ogni operazione.

Output relativi alle memorie cache

`Access time (ns)`

- Tempo per accedere alla memoria e ottenere il dato richiesto, considerando anche la valutazione di *hit* o *miss* nella memoria.

`Cycle time (ns)`

- Tempo minimo che deve intercorrere tra due operazioni di memoria.

`Total dynamic read energy per access (nJ)`

- Energia consumata dalla memoria e dalla logica di supporto ad essa per eseguire una operazione di lettura.

`Total dynamic write energy per access (nJ)`

- Energia consumata dalla memoria e dalla logica di supporto ad essa per eseguire una operazione di scrittura.

Gli ultimi due valori di energia sopra presentati tengono conto di tutte le micro-operazioni necessarie per espletare i comandi di lettura o scrittura richiesta. Alcune di queste operazioni sono:

1. Tempo ed energia per la selezione del corretto banco di memoria;
2. Tempo ed energia per la selezione del *set* all'interno della cache;
3. Tempo ed energia per la valutazione dei *tag* per ogni linea di cache all'interno del set selezionato; Tempo ed energia per lo scaricamento/caricamento dal bit-array del dato cercato

`Total leakage power of a bank (mW)`

- Potenza di *leakage* dissipata da ogni banco durante il regolare funzionamento

`Cache height x width (mm)`

- Dimensione in millimetri del modulo di memoria cache, compreso di banchi e logica di supporto alla memoria.

Output relative alle memorie principali

`t_RCD (ns)`

- *Row to Column command delay*, il tempo che il dato impiega ad arrivare dalle *bitline* ai *row* buffers

`t_RAS (ns)`

- Minimo tempo per il quale una riga di memoria deve stare aperta per consentire lo scaricamento del dato

`t_CAS (ns)`

- Tempo che intercorre tra l'attivazione della Colonna e l'arrivo del dato in uscita

`t_RP (ns)`

- Tempo per caricare una riga di DRAM

`t_RRD (ns)`

- Tempo che intercorre tra il momento in cui viene dato il comando alla riga a quando effettivamente la riga si attiva

`t_RC (ns) = tRAS + tRP`

- *Row "cycle" time*

`Activation energy (nJ)`

- Energia consumata per la selezione e l'attivazione del banco di memoria dove sarà selezionato il dato da leggere/scrivere

`Read energy (nJ)`

- Energia consumata per eseguire l'operazione di lettura vera e propria, ovvero lo scaricamento del dato dal *bit-array* nei *row buffers* sottostanti il banco selezionato (quindi l'energia per caricare i *sense amplifiers*)

`Write energy (nJ)`

- Energia consumata per eseguire l'operazione di scrittura vera e propria, ovvero la scrittura dentro ai row buffers sottostanti il banco selezionato e il trasferimento del dato dentro al bit-array corretto.

`Precharge energy (nJ)`

- Energia consumata per eseguire l'operazione di *precharge* della *bitline* selezionata, operazione preliminare alla scaricamento del dato verso o da i *row buffers*.

`DRAM die width (mm)`

`DRAM die height (mm)`

- Dimensioni in millimetri di ogni die di DRAM

I parametri sopra elencati devono essere oculatamente selezionati per la valutazione di *delay* ed energia consumata per le operazione di *read* o di *write*. Non tutte le componenti sopra infatti entrano in gioco ad ogni operazione, ma variano in relazione ad alcuni eventi
quali:

1. il banco di DRAM dove si deve leggere/scrivere è già attivato in quanto la precedente operazione di memoria aveva coinvolto quel banco stesso;
2. il dato da scrivere/leggere è presente o meno all'interno del *row buffers* relativo al banco di DRAM coinvolto nell'operazione.

In base alle condizioni sopra identificate, possiamo evidenziare quattro casi principali per i quali valutare quali componenti effettivamente entrano in gioco per la valutazione del delay e dell'energia consumata.

I. HIT - Si verifica quando il dato interessato dall'operazione di lettura/scrittura è già presente nel *row buffer* del relativo banco di memoria, e in più il banco interessato era stato già attivato dalla operazione di lettura/scrittura precedente a quella presa in esame. In questo caso il *delay* di operazione di lettura/scrittura sarà t_{CAS} mentre l'energia consumata dalla operazione di lettura/scrittura sarà $Read_{Energy}$ o $Write_{Energy}$
II. SEMI-HIT - Si verifica quando il dato interessato dall'operazione di lettura/scrittura è già presente nel *row buffer* del relativo banco di memoria, ma il banco interessato nell'operazione deve essere preventivamente selezionato poiché non interessato dall'operazione di memoria precedente a quella presa in esame. Si avrà che il delay di operazione di lettura/scrittura sarà $t_{CAS} + t_{RAS}$ mentre

l'energia consumata dalla operazione di lettura/scrittura sarà $Read_{Energy} + Activation_{Energy}$oppure $Write_{Energy} + Activation_{Energy}$

III. SEMI-MISS - Si verifica quando il banco di memoria contenente il dato da dover prelevare per l'operazione di lettura/scrittura risulta già attivato poiché utilizzato dall'operazione di memoria precedente, ma tale dato non risulta essere nel *row buffer* e dovrà essere scaricato dal *bit-array*. Il *delay* di operazione di lettura/scrittura sarà $t_{RP} + t_{RCS} + t_{CAS}$ mentre l'energia consumata dalla operazione di lettura/scrittura equivarrà $Read_{Energy} + Precharge_{Energy}$ oppure $Write_{Energy} + Precharge_{Energy}$

IV. MISS - Si verifica quando si deve preventivamente selezionare il banco ove andare a prelevare il dato interessato dall'operazione di memoria e tale dato non è presente nel *row buffer* di tale banco, rendendo così necessario il suo scaricamento preventivo dalla *bit-array*. In questo caso il delay di operazione di lettura/scrittura sarà $t_{RP} + t_{RCS} + t_{CAS} + t_{RAS}$ e l'energia invece: $Read_{Energy} + Activation_{Energy} + Precharge_{Energy}$ oppure $Write_{Energy} + Activation_{Energy} + Precharge_{Energy}$

Cacti-3DD

CACTI-3DD è una recente evoluzione di CACTI volta alla modellazione di memorie 3D-DRAM *Stacked*. Come si legge in (24) questa versione del potente strumento di modellazione di memorie rende possibile lo studio approfondito di questa recente e innovativa tecnologia, già ampiamente presentata nell'introduzione, e restituisce informazioni dettagliate inerenti ogni operazione di memoria, lettura e scrittura, energie consumate durante le operazioni e dal bus interno costituito dalle TSV.

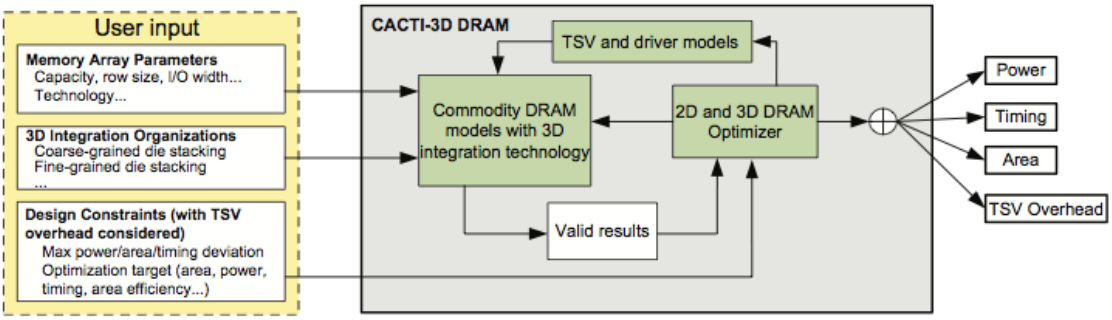

Figura : Diagramma a blocchi del framework di CACTI-3DD

In Figura è presentato lo schema a blocchi del *framework* di CACTI-3DD, che si differenzia da quello classico di CACTI per l'integrazione di una tecnologia 3D *Stacked* per la modellazione dei moduli di memoria e l'integrazione e la modellazione dei bus TSV.

Il software, oltre a tutti i parametri già ampiamente argomentati precedentemente permette la specifica di alcuni altri valori strettamente legati alla natura stessa della memoria 3D.

I parametri da dover specificare per la modellazione di una memoria 3D *Stacked*, in aggiunta a tutti quelli relativi ad una memoria principale DRAM, sono:

```
cache type "3D memory"
```

con il quale si forzerà il software alla simulazione di una memoria 3D *Stacked*,

```
stacked die count <value>
```

con il quale si potrà specificare il numero di die *stacked* che si desidera simulare ed infine

```
partitioning granularity <value (0 - 1)>
# 0 - WIDE-3D # 1 - TRUE-3D
```

Questo è forse il parametro più importante su cui porre l'attenzione. La scelta di questo valore permettere la modellazione di una DRAM 3D *Stacked* nelle sue due principali forme, WIDE-3d oppure TRUE-3D, individuate e presentate con le relative fonti nell'introduzione del documento e di cui si riporta un modello alternativo in Figura .

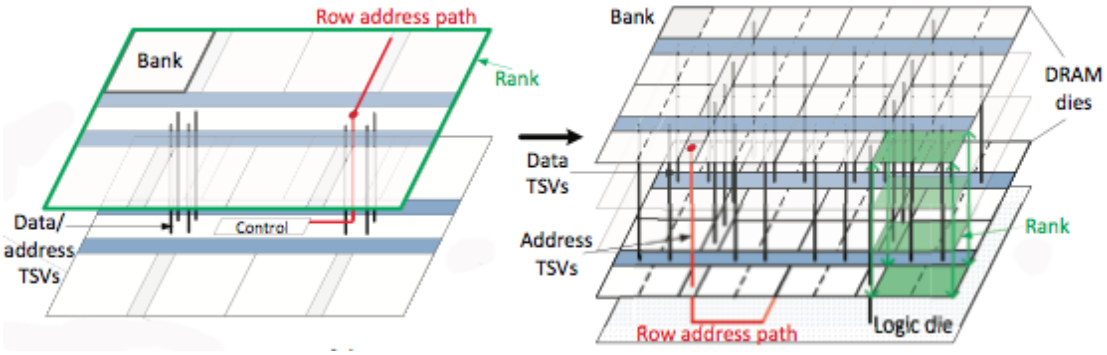

Figura : Modello architetturale per WIDE-3D e TRUE-3D Stacked DRAM

Ovviamente utilizzando la configurazione TRUE-3D si otterranno migliori prestazioni in termini di delay per ogni tipo di operazione a costo di un maggior consumo di energia dovuto alla fitta rete di bus TSV che collegano banchi di die differenti e appartenenti allo stesso rango all'unità di elaborazione, al posto di avere un unico *stripe bus* centrale soggetto ovviamente a problemi di competizione.

CACTI-3DD ancora non supporta la modellazione dei singoli banchi *stacked*, ulteriore ottimizzazione del paradigma TRUE-3D, continuando ad utilizzare il classico modello a banchi planari.

Il modello del *bank-stacked die* è riportato in Figura

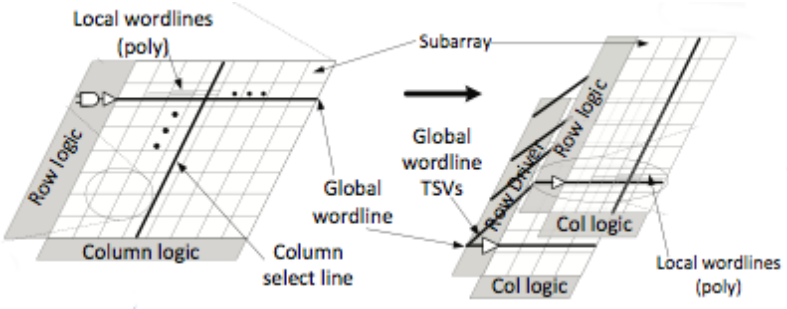

Figura : Evoluzione dal planar-bank allo Stacked-bank

In questa configurazione ogni singolo banco di memoria risulta essere a sua volta *stacked* in modo tale da accelerare il recupero dei dati dai *bit-array* verso i *row buffers*. In questo caso i comandi di selezione di *WordLine* e *ColumnLine* vengono estesi ad ogni *sub-bank stacked* in modo tale che lo scaricamento/caricamento del dato nei/verso i *row buffers* possa avvenire in parallelo. Notare che anche la comunicazione tra i vari *sub-banks* avviene per mezzo di TSV.

Wind River Simics

Wind River SIMICS (25) è un *full system simulator* che mette a disposizione diversi strumenti per la modellazione di architetture definite dall'utente da poter integrare, testare e debuggare in un ricco set di sistemi moderni pre-configurati nel sistema.

Con l'evolversi della tecnologia, i sistemi moderni sono diventati sempre più complessi: esistono sia sistemi con processori *general purpose multi-core* in grado di supportare molteplici sistemi operativi sia sistemi HPC con centinaia di *core* fisici e migliaia di *core* virtuali con enormi e complessi sistemi di memoria fisici o *cloud-based.*

Innanzi a questa moltitudine di complessi sistemi diventa quasi impossibile riuscire a ideare, modellare, realizzare, integrare e testare una nuova tecnologia.

Wind River SIMICS permette dunque la simulazione di ogni aspetto legato ad un sistema per quanto complesso possa essere, dal più alto livello costituito dalle applicazione utente al livello più basso costituito dalle unità di memorizzazione volatili più vicine al processore: le cache.

Grazie a questo la definizione di una nuova architettura da integrate ad un *system target* già esistente diventa molto più accessibile. In Figura possiamo vedere uno schema ad alto livello del framework di simulazione offerto da SIMICS.

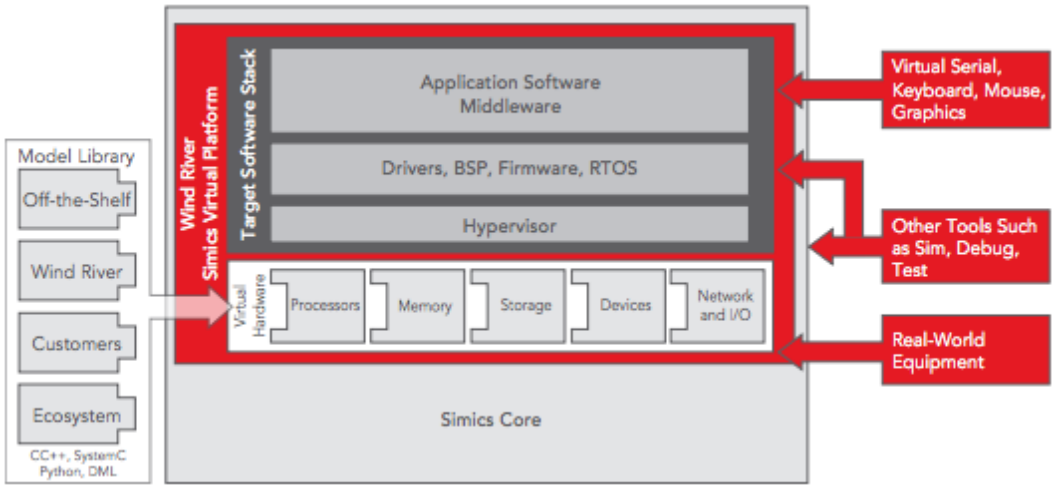

Figura : Framework offerto da Simics

Come si nota dallo schema sopra, SIMICS permette la selezione di una architettura target sopra la quale poter installare un sistema operativo scelto dall'utente. Alcune di queste architetture target sono:

- PowerPC
- AMD
- Intel
- MIPS
- M68K
- SPARC

Al momento della definizione dell'architettura si può scegliere (se l'architettura lo prevede ovviamente) il numero di *core* fisici da utilizzare, la quantità totale di memoria principale del sistema, *device* di rete, supporti rimovibili quali CD-ROM, DVD-ROM e altri.

Quello che più tuttavia interessa i progettisti è la possibilità di utilizzare moduli aggiuntivi, pre-configurati o *user-defined*, per la simulazione di aggiuntive architetture da voler integrare col sistema.

Uno dei più interessanti moduli che SIMICS offre per la modellazione di componenti hardware è il modulo **g-cache**. Questo modulo simula il funzionamento classico di una cache di memoria, privata o condivisa, integrando al suo interno protocolli già consolidati come il protocollo MESI per la gestione della consistenza dei dati in caso di architettura che utilizzano cache condivise in ambienti multi-core (Si veda a tal proposito il paragrafo relativo al protocollo MESI e alla gestione della coerenza nella 3D-DRAM come LLC).

Attraverso questi moduli è possibile creare un file di configurazione che descriva l'architettura di memoria che si vuole integrare e simulare con il target selezionato, semplicemente collegando in cascata a proprio piacimento i suddetti moduli. Nulla vieta di poter modellare un modulo per la simulazione della memoria principale, da collegare in ultimo stadio alle cache, per simulare una complessa e più completa architettura di memoria.

Simulare una architettura di memoria è utile per valutare le performance in termini di ritardi e consumo di energia. Ogni volta che il target simulato genera una operazione di memoria, questa viene processata e viene fatta "gestire" dall'architettura che lo sviluppatore ha definito, registrando, stadio per stadio, lo stallo che l'unità di elaborazione ha sperimentato per il completamento dell'operazione di memoria stessa.

Lo studio qui presentato ha riguardato la definizione di una architettura di memoria attraverso la creazione di nuovi moduli, la creazione di un target, l'installazione di un sistema operativo sopra di esso, la scelta di applicazioni utente da eseguire su questo sistema e la raccolta di statistiche inerenti le performance ottenute dall'architettura così come definita.

Più in dettaglio le operazioni da eseguire per la definizione di un target e la simulazione di una architettura sono le seguenti:

1. Scelta di un target per il sistema (per le nostre simulazioni si è usata una architettura x86_64)
    a. numero di *core* fisici
    b. limite di memoria principale
2. Creazione di una macchina virtuale e installazione di un sistema operativo su di essa;
3. Conversione della macchina virtuale nel formato proprietario SIMICS in modo tale da poter essere avviata sul target prima selezionato;
4. Scelta, installazione ed esecuzione di programmi applicativi sulla macchina virtuale volti al *testing* dell'architettura da simulare (per lo più si tratta di programmi applicativi *benchmark*);
5. Creazione di un checkpoint, ovvero una "istantanea" dell'intero sistema: contenuto della RAM, contenuto dell'Hard Disk e delle altre memorie da poter poi avviare di nuovo insieme ai moduli aggiuntivi per il *testing* del sistema;
6. Creazione del file di configurazione per la descrizione dell'architettura modellata dallo sviluppatore che si vuole simulare nel regolare funzionamento del sistema;

7. Avvio del sistema simulato con il checkpoint sopra menzionato e con il file di configurazione precaricato per il testing della nuova architettura;
8. Raccolta delle statistiche per la valutazione della architettura modellata.

La descrizione dettagliata del *target* utilizzato, del sistema operativo installato nella macchina virtuale, dei *benchmark* utilizzati e dei moduli creati per la definizione delle nuove architetture studiate in questo documento verrà presentata nel capitolo dedicato alle simulazioni e alla raccolta e valutazione dei risultati.

Benchmarks Suites

In questo capitolo verranno brevemente esaminate le suite di *benchmark* utilizzate per lo *stress test* delle architetture presentate precedentemente.

Le suite utilizzate sono per i test sono state due:

1. SPEC CPU2006: suite del famoso consorzio SPEC specializzato nella definizione di standard per la valutazione di sistemi informatici e nella creazione, modifica e raccolta di applicativi volti al test di CPU e sottosistemi di memoria;
2. OLTP: suite di applicativi, caratterizzati da un elevato *footprint* di memoria, rivolti al test di architetture *Cloud* e HPC.

Spec CPU2006

SPEC CPU2006 è una suite di *benchmark* che mette a disposizione una vasta gamma di applicativi per il calcolo di parametri di valutazione delle performance per un ampio set di diversi tipi di architetture hardware. Ogni *benchmark* è caratterizzato da un differente *workload* parzialmente configurabile, che spazia dalle poche centinaia di MB ai $2 \div 3$ GB e da un carico computazionale tipico di applicazioni in ambito scientifico.

La suite CPU2006 è distribuita in singoli pacchetti di codice sorgente in modo tale da permettere all'utilizzatore la scelta di *workload* da voler utilizzare e la loro modifica per un eventuale adattamento al proprio sistema da voler simulare. I *workload* disponibili sono 3, e vengono riportati di seguito ordinati secondo la complessità computazione e la quantità di memoria utilizzata:

1. Test
2. Train
3. Reference

Di seguito viene riportato un sottoinsieme di questi *benchmark* utilizzati nelle simulazioni e una loro breve descrizione.

`401.bzip2`

- Area di applicazione : compressione
- Linguaggio di programmazione : `C`
- Descrizione: *benchmark* basato sull'algoritmo bzip v.1.0.3, di Julian Seward. Unica differenza tra il classico algoritmo di compressione e questa versione adattata dal consorzio SPEC riguarda il fatto che in questa versione le uniche operazioni di I/O effettuato sono quelle che riguardano la lettura del file di input;

`403.gcc`

- Area di applicazione: C compiler
- Linguaggio di programmazione: `C`
- Descrizione: *benchmark* basato sul celebre compilatore gcc v.3.2, di Richard Stallman. Il *benchmark* genera codice per un processore AMD Opteron;

`429.mcf`

- Area di applicazione: ottimizzazione combinatoria
- Linguaggio di programmazione: `C`
- Descrizione: *benchmark* basato su un algoritmo per la gestione del traffico utilizzato in molte applicazioni commerciali particolarmente rivolte allo smistamento del trasporto pubblico;

`445.gobmk`

- Area di applicazione: intelligenza artificiale (Go)
- Linguaggio di programmazione: `C`

- Descrizione: *benchmark* basato sul gioco del Go; simula una partita tra due giocatori dotati di intelligenza virtuale;

`456.hmmer`

- Area di applicazione: ricerca di sequenze genetiche
- Linguaggio di programmazione: `C`
- Descrizione: Analisi di sequenze proteiche utilizzando il profilo dei modelli nascosti di Markov;

`458.sjeng`

- Area di applicazione: intelligenza artificiale (scacchi)
- Linguaggio di programmazione: `C`
- Descrizione: *benchmark* che simula una partita di scacchi tra due giocatori di alto livello, giocando anche alcuni varianti del gioco stesso;

`462.libquantum`

- Area di applicazione: Physics / Quantum Computing
- Linguaggio di programmazione: `C`
- Descrizione: benchmark che simula un calcolatore quantistico che esegue l'algoritmo di fattorizzazione in tempo polinomiale di Shor;

`464.h264ref`

- Area di applicazione: compressione video
- Linguaggio di programmazione: `C`
- Descrizione: *benchmark* implementato sull'emergente algoritmo di compressione H.264/AVC. L'algoritmo codifica uno *stream* video utilizzando due set di parametri di compressione differenti.

`473.astar`

- Area di applicazione: algoritmi per la ricerca di percorsi
- Linguaggio di programmazione: `C++`
- Descrizione: il *benchmark* fa uso della libreria Pathfinding per la ricerca di percorsi all'interno di mappe 2D, incluso il noto algoritmo A*;

`471.omnetpp`

- Area di applicazione: simulazione di eventi discreti
- Linguaggio di programmazione: C++
- Descrizione: *benchmark* che utilizza il simulatore OMNet++ per modellare una grande rete Ethernet all'interno di un campus;

410.bwaves

- Area di applicazione: fluidodinamica
- Linguaggio di programmazione: FORTRAN
- Descrizione: *benchmark* che calcola il flusso tri-dimensionale laminare viscoso transonico transiente;

416.gamess

- Area di applicazione: chimica quantistica
- Linguaggio di programmazione: FORTRAN
- Descrizione: *benchmark* che implementa un vasto insieme di calcoli quantistici in ambito chimico;

433.milc

- Area di applicazione: cromo-dinamica fisico-quantistica
- Linguaggio di programmazione: C
- Descrizione: *benchmark* che esegue un algoritmo per la generazione di campi di forza nella teoria dei campi reticolari con quark dinamici;

434.zeusmp

- Area di applicazione: fluidodinamica computazionale
- Linguaggio di programmazione: FORTAN
- Descrizione: *benchmark* basato su un algoritmo sviluppato dal Laboratory for Computational Astrophysics per la simulazione di fenomeni astrofisici;

435.gromacs

- Area di applicazione: biochimica, dinamiche molecolari
- Linguaggio di programmazione: C, FORTRAN

- Descrizione: il *benchmark* simula un algoritmo per le dinamiche molecolari, ovvero simula le equazioni Newtoniane per il moto di centinaia di milioni di particelle.

436.cactusADM

- Area di applicazione: fisica, relatività generale
- Linguaggio di programmazione: C, FORTRAN
- Descrizione: il *benchmark* risolve le equazioni di evoluzione di Einstein utilizzando il metodo numerico delle cavalline-sfilzate;

437.leslie3d

- Area di applicazione: fluidodinamica
- Linguaggio di programmazione: FORTRAN
- Descrizione: il *benchmark* esegue un algoritmo nell'ambito della fluidodinamica computazionale.

444.namd

- Area di applicazione: biologia
- Linguaggio di programmazione: C
- Descrizione: il *benchmark* simula grandi sistemi biomolecolari.

OLTP

OLTP è una suite di *cloud benchmarks* rivolti al test di architetture server *cloud* e HPC. La suite mette a disposizione molteplici parametri d configurazione per una profonda definizione dei *workloads*, come ad esempio numero di client connessi ad un server, il numero di richieste che questi client effettuano e l'intervallo a cui queste vengono eseguite, permettendo inoltre la configurazione del lato *back-end* del sistema *cloud*.

Di seguito viene proposta la lista dei *benchmarks* utilizzati e una loro breve descrizione.

AuctionMark

Il *benchmark* simula caratteristiche di *workload* di un sito di aste e annunci web.

Epinions

Il *benchmark* simula il traffico del noto sito Epinions.com, portale che raccoglie valutazioni di prodotti da parte di clienti consumatori.

TATP

Il *benchmark* simula un sistema di locazione di locazione di chiamate utilizzato tipicamente dai provider di servizi di tele-comunicazione.

SEATS

Il benchmark simula un sito di prenotazione di voli aerei, comprese le ricerche dei voli e la loro prenotazione.

SIMULAZIONI E ANALISI DEI RISULTATI

Dopo aver esposto in maniera esaustiva non solo le architetture esaminate ma anche tutta la strumentazione utilizzata per la modellazione delle stesse è necessario voltare pagina e passare alla trattazione dell'argomento forse principale di questo studio: l'esposizione e analisi degli output prodotti dalle simulazioni.

Come già trattato precedentemente gli strumenti di simulazione sono stati principalmente due: Cacti e Simics. Il primo utilizzato per la modellazione e il secondo per la messa in commercio dei componenti. Si partirà quindi descrivendo le varie architetture modellate con Cacti e si discuteranno le motivazioni che hanno portato a scegliere, per la prova con Simics, un'architettura piuttosto che un'altra.

Simulazioni e analisi dei risultati ottenuti con Cacti

I seguenti sotto-paragrafi saranno dedicati ciascuno alla scelta della giusta variante per ogni componente del sistema.

Baseline

Preliminarmente è utile stabilire un termine di paragone tramite il quale poter effettuare il giusto tipo di comparazioni per decidere, al termine del lavoro, se la nuova architettura risulta migliore o peggiore della precedente e, cosa da non sottovalutare, in che percentuale essa risulta esserlo.

Nel caso in esame, ovviamente, la baseline è un tipo di memoria principale realizzato tramite tecnologie già affermate sul mercato che servirà, come già accennato, quale termine di paragone con tutte le soluzioni in analisi che sono state esposte precedentemente in questo documento.

La baseline che è stata scelta, in primis, è sicuramente una baseline molto generosa. Infatti si è presupposto una memoria 2D-DRAM, realizzata secondo gli standard DDR3, dalla capacità infinita affiancata da un TLB di capacità anch'essa

infinita. Questa scelta, sebbene irrealizzabile e dunque apparentemente in contrasto con la fisica realizzazione enunciata precedentemente, è da giustificarsi poiché consente di ottenere un sistema simulato nel quale non accadono *page faults* se non nel primo accesso ad una certa locazione di memoria (*Cold Miss*).

Nonostante la irrealizzabilità fisica della memoria si è deciso di considerarla composta, a basso livello, da banchi provenienti da tecnologie DRAM esistenti. Cacti è stato dunque utilizzato per l'analisi di questi banchi e si è infine effettuata una scelta tra di essi in maniera da avere la DRAM quanto più veloce possibile.

I tipi memoria che sono stati analizzati con Cacti sono riportati nella tabella seguente.

Tabella : Tipi di memoria analizzati per la determinazione della Baseline

Nome Simbolico	*Grandezza (GB)*	*Numero di Banchi*	*Grandezza della pagina*
2D-DRAM_8GB_8Banks_4KBpP	8	8	$4\ KB$
2D-DRAM_8GB_16Banks_4KBpP	8	16	$4\ KB$
2D-DRAM_16GB_8Banks_4KBpP	16	8	$4\ KB$
2D-DRAM_16GB_16Banks_4KBpP	16	16	$4\ KB$

Si noti innanzitutto come la dimensione di una singola pagina è stata mantenuta costante a 4 KB in tutte le configurazioni in analisi. Questa scelta, a prima vista non giustificata, ha le sue ragioni nella storia passata dell'architettura dei calcolatori: tutti gli hard disk tradizionali, con i quali si presuppone questa memoria 2D dovrà convivere hanno infatti, lato software, l'assistenza del sistema operativo che impone in qualche modo la dimensione della pagina a tale grandezza. Naturalmente questa non è una vera e propria imposizione in quanto l'utente finale può sempre decidere di modificare tale valore sebbene il caso più comune rimanga quello di unità di trasferimento minimo tra disco e memoria pari proprio a 4 KB.

Un'altra scelta che può risultare strana è, visto che si è appena parlato di DRAM infinita, quella di aver imposto una dimensione finita alla DRAM simulata. Questa scelta è dualmente giustificata. Infatti non solo il simulatore Cacti giustamente non permette di inserire `+Infinity` come valore per `size` ma anche è soprattutto quello che si considera

è una memoria DRAM, planare, infinita in grandezza ma nel dettaglio composta da banchi la cui efficienza è la medesima di quella che avrebbero se posti in memorie DRAM da grandezza finita. I risultati che evidenziano quale sia la migliore baseline sono esposti di seguito.

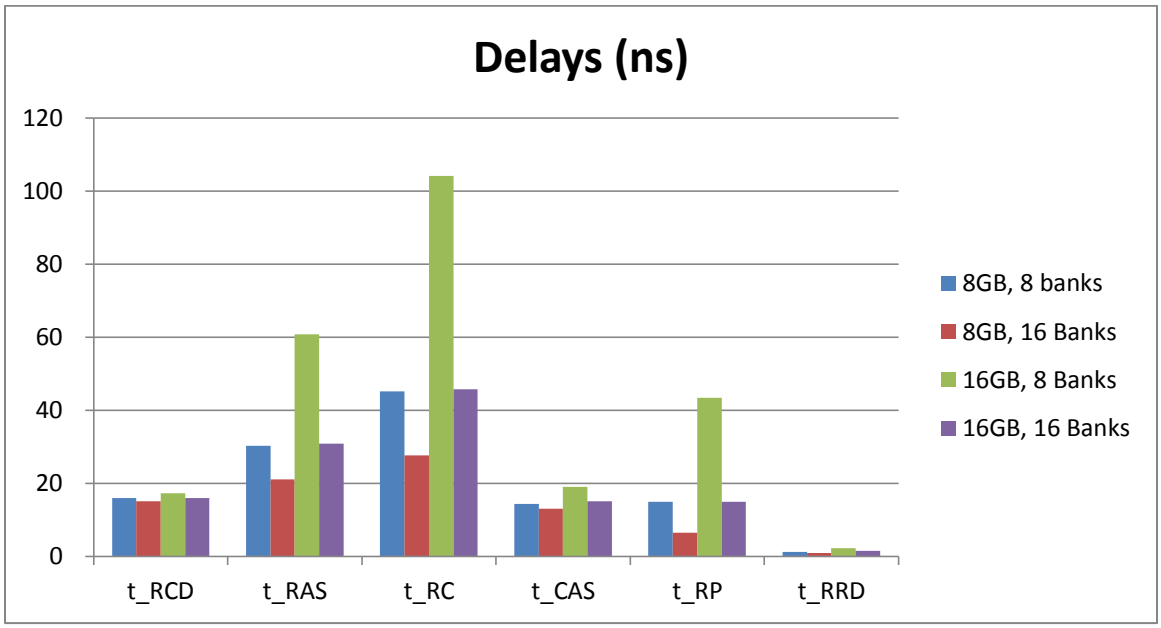

Grafico : Latenze (ns) per le componenti di accesso alla DRAM

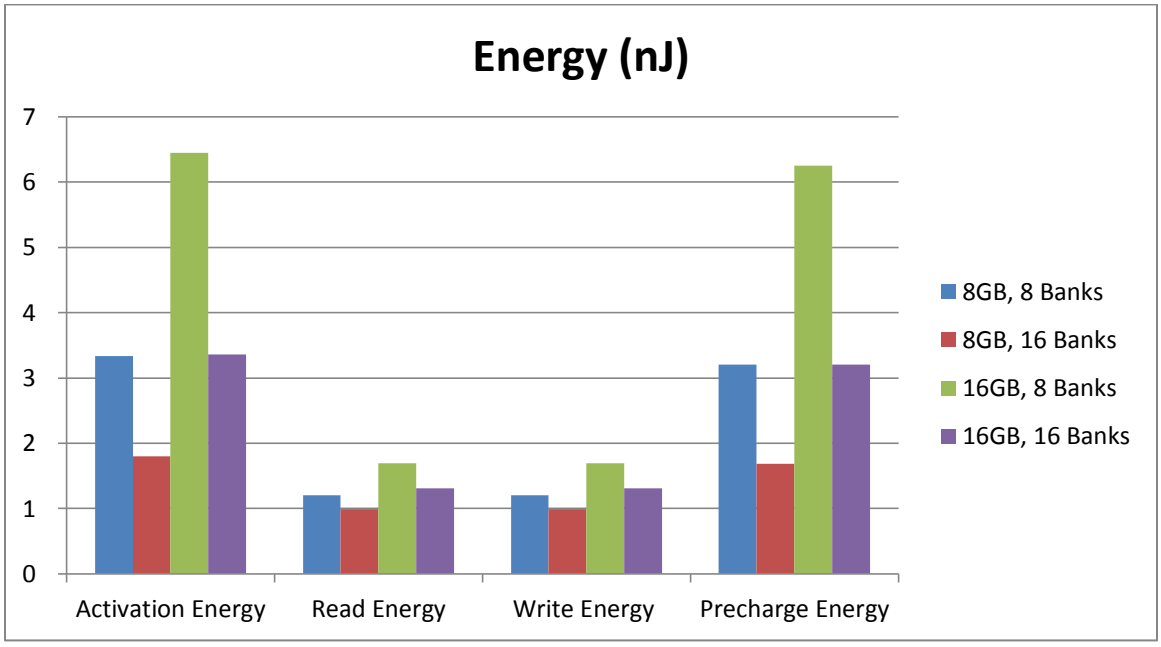

Grafico : Energia (nJ) consumata per un singolo accesso in memoria

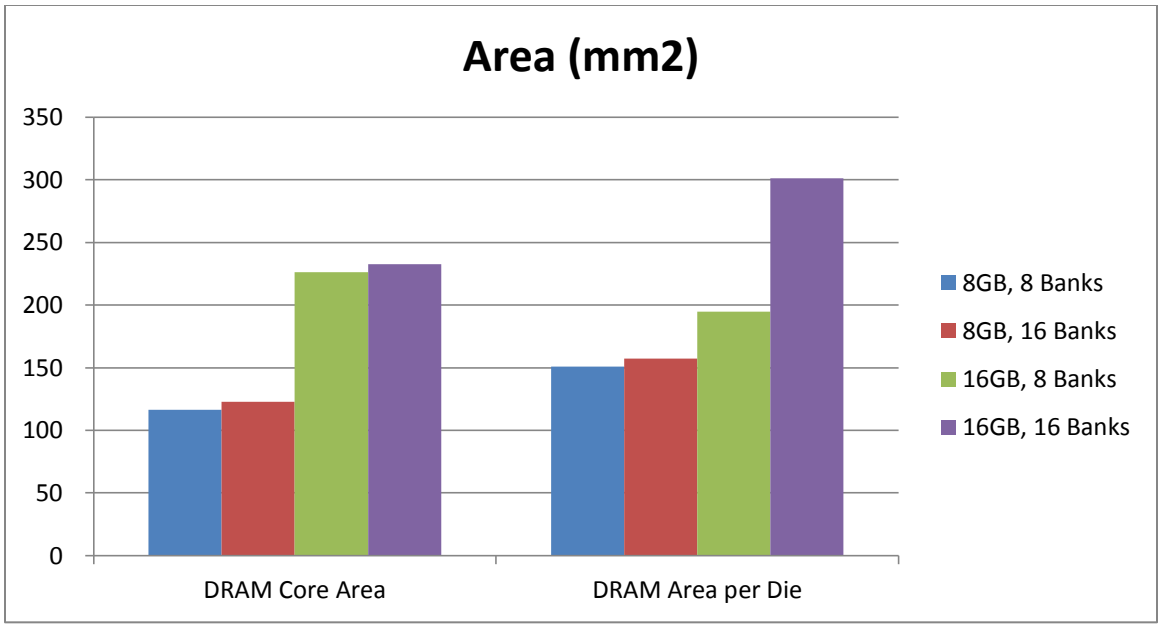

Grafico : Area (mm^2) delle varie memorie

I risultati esposti dai grafici non lasciano alcun dubbio sulla scelta da effettuare: la combinazione di 8GB di memoria organizzati in 16 banchi è quella per cui tutte le varie componenti temporali sono minime e dunque anche le componenti composte, somma delle componenti semplici, risulteranno minime. Non solo, perché anche da un punto di vista del consumo energetico la suddetta memoria risulta essere di gran lunga più efficiente delle altre. Unico termine di paragone secondo il quale la configurazione non risulta eccellente è quello relativo all'area utilizzata su chip. L'*overhead*, dovuto principalmente alla logica aggiuntiva necessaria alla gestione di un maggior numero di banchi, è tuttavia trascurabile essendo al più di 20 mm^2.

Risulta semplice dunque stabilire come la memoria che risulta essere migliore per la modellazione della baseline sia quella il cui nome simbolico riportato in Tabella è 2D-DRAM_8GB_16Banks_4KBpP.

TLB

Ognuna delle architetture esposte in precedenza fa uso di un modulo di traduzione basato sulla CMM. Per tale motivo è importante definire quale tipo di TLB utilizzare nell'architettura stessa.

Come già accennato si è cercato di migliorare notevolmente il TLB precedentemente progettato dagli autori di (13) introducendo per esso un'associatività ad insiemi ed espandendone le dimensioni fino a 2048 entrate. Primo problema dunque da tenere in considerazione è il fatto che passare da un TLB a 32 o 64 entrate per CPU ad uno che possiede 2048 entrate per CPU è quello che il TLB stesso occuperà nella nuova configurazione un'area enorme. Tuttavia se si ricorda che la CMM è costruita mediante stacking delle componenti risulta possibile posizionare il TLB direttamente sopra ognuno dei core al layer1 dell'architettura oppure, se si utilizza una delle architetture come 3D-DRAM LLC, posizionarlo laddove risiede nei processori odierni la cache di terzo livello.

Nell'APPENDICE B: File Sorgenti è riportato il codice dell'intestazione del modulo del TLB. Da enfatizzare è il fatto che i nomi delle funzioni che lo compongono (e quindi i nomi della API che esso espone) sono rimasti invariati per retro-compatibilità.

I TLB simulati con Cacti sono quelli riportati nella tabella seguente.

Tabella : Tipi di TLB simulati con Cacti

Nome Simbolico	*Entrate*	*Associatività*
TLB_Fully_512Entries	512	Fully
TLB_Fully_1024Entries	1024	Fully
TLB_Fully_2048Entries	2048	Fully
TLB_WA4_512Entries	512	4
TLB_WA4_1024Entries	1024	4
TLB_WA4_2048Entries	2048	4
TLB_WA8_512Entries	512	8
TLB_WA8_1024Entries	1024	8
TLB_WA8_2048Entries	2048	8
TLB_WA16_512Entries	512	16
TLB_WA16_1024Entries	1024	16
TLB_WA16_2048Entries	2048	16

La divisione dei vari TLB è semplice e lineare: le dimensioni variano da 512 a 2048 saltando in potenze di 2 mentre l'associatività è completa oppure varia da 4 a 16 sempre saltando in potenze di 2.

I grafici seguenti mostrano le performance ottenute dai TLB appena esposti.

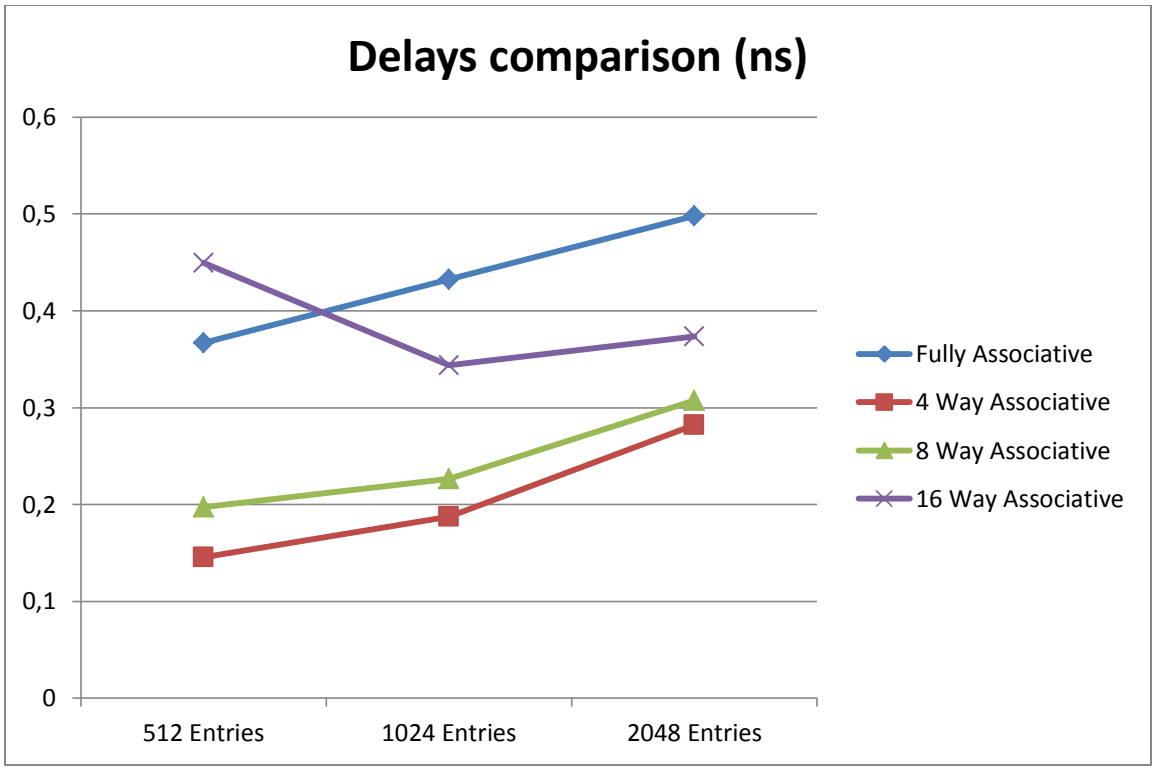

Grafico : Comparazione Latenze TLB

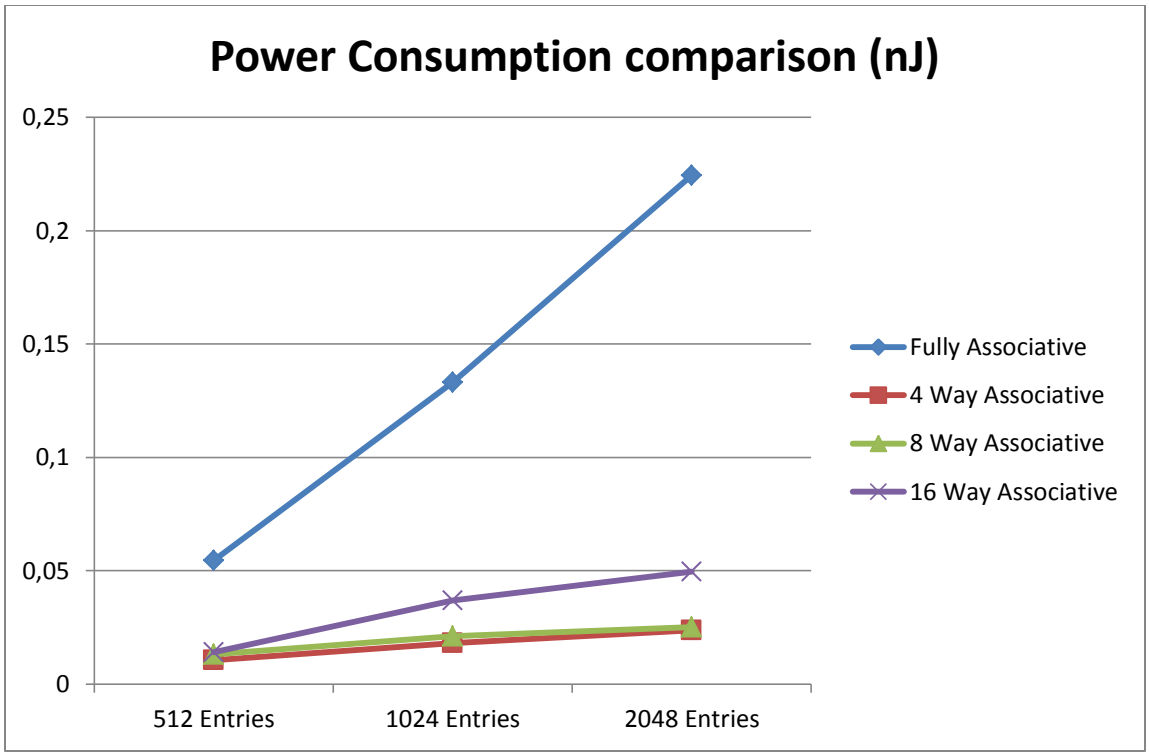

Grafico : Comparazione Energia Consumata dal TLB

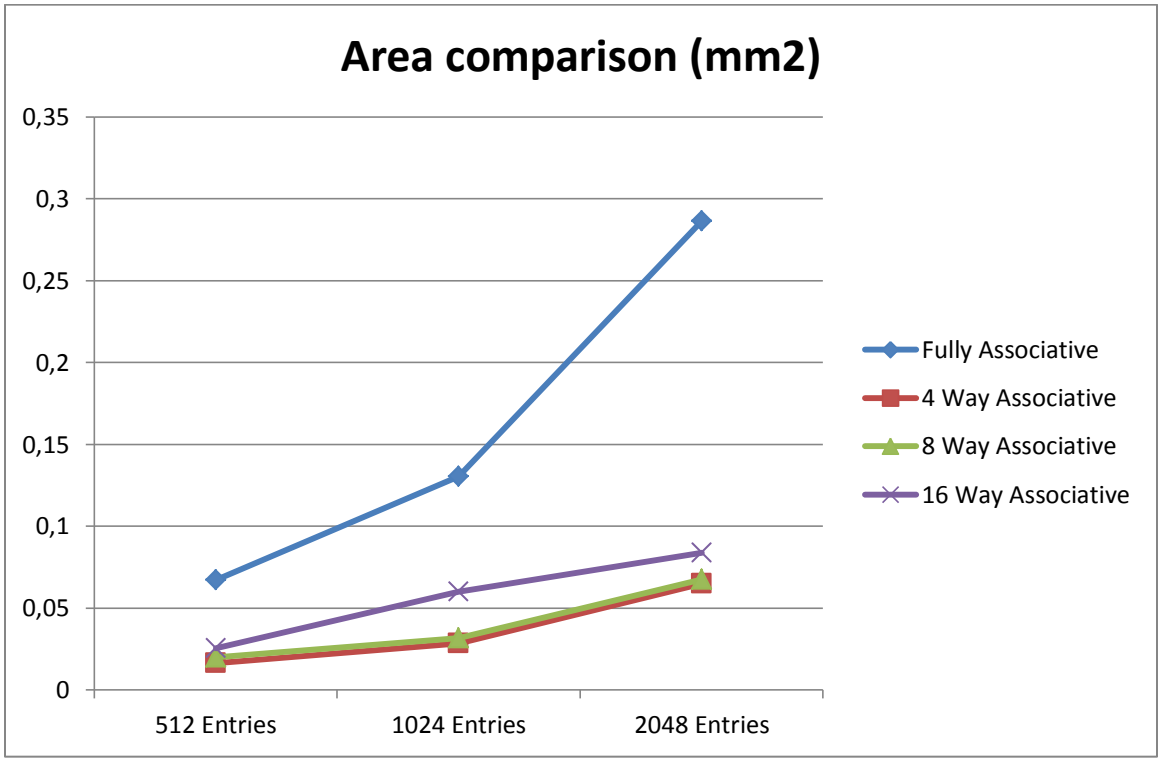

Grafico : Comparazione dell'area occupata dal TLB

Come è facile convincersi dall'esame dei grafici sopra riportati, la scelta di un TLB completamente associativo mantenendo un alto numero di entrate è una scelta che definire sbagliata sarebbe un eufemismo. Con l'esclusione della versione 16-*way associative* nel grafico relativo ai *delay* infatti, il TLB fully associative risulta il peggiore sotto ogni punto di vista.

Proprio continuando dal TLB 16-*way* è anche qui facile notare come per lo meno per quanto riguarda i *delay* i tempi ottenuti sono molto alti. Questo esclude dunque anche un TLB a 16 *way* dai possibili candidati.

Le rimanenti combinazioni invece risultano tutte molto valide e vicine tra loro in termini di *delay*, energia ed area. Al fine di mantenere comunque molto alta la *hit rate* si è deciso di escludere tutte le varianti da 512 entry e mantenere quindi attive solo le seguenti:

- 1024 entrate, 4-*way associative*
- 1024 entrate, 8-*way associative*
- 2048 entrate, 4-*way associative*
- 2048 entrate, 8-*way associative*

SRAM

Un'ulteriore componente della CMM è la SRAM il cui scopo è quello di assicurare una veloce traduzione dell'indirizzo nel caso in cui il TLB dia miss. La SRAM dotata di logica aggiuntiva, al fine di realizzare quello che nei paragrafi precedenti è stato denominato *index-trick,* ha la necessità di essere abbastanza capiente da contenere un numero di entrate pari al numero di pagine totalmente residente in memoria. Supponiamo, come già calcolato dagli autori di (13) che la dimensione ottimale per una pagina fisica sia di 32KB. Il numero di entrate totali necessari nella SRAM sarà dunque:

$$\#SRAM_{entries} = \frac{MEM_{Size}}{32768}$$

Nello studio qui presentato la dimensione della memoria presa in considerazione oscilla tra 8 e 32 GB. Il numero di entrate quindi sarà $\frac{8 \cdot 2^{30}}{2^{15}} = 8 \cdot 2^{15} = 262144$ per la

versione a 8 GB, $\frac{16 \cdot 2^{30}}{2^{15}} = 16 \cdot 2^{15} = 524888$ per la versione a 16 GB e infine $\frac{32 \cdot 2^{30}}{2^{15}} = 32 \cdot 2^{15} = 1048576$ per la versione a 32 GB.

Ricordando inoltre che la struttura di una singola entrata nella SRAM, già denominata CMM/TAG Location, è composta come segue

Tabella : Struttura finale di una entrata in SRAM

Segment Address **12 ÷ 14 bit**	Virtual TAG **10 bit**	Protection **2 bit**

Abbiamo che la dimensione totale della memoria SRAM oscilla da un minimo di

$$\left\lceil \frac{12 + 10 + 2}{8} \right\rceil \cdot 262144 = 786432\ byte = 784\ KB$$

Ad un massimo di

$$\left\lceil \frac{14 + 10 + 2}{8} \right\rceil \cdot 1048576 = 4194304\ byte = 4\ MB$$

Da considerare comunque è anche il fatto che, non essendo in questa implementazione ancora presente il traduttore dell'indirizzo di segmento la memoria effettivamente istanziata durante le simulazioni risulta essere inferiore a quella prevista.

Anche per la SRAM si è utilizzato CACTI e ne sono state simulate diverse versioni per una singola dimensione (ossia, fissato il numero di entrate, sono state provate diverse varianti). La tabella presente mostra, in maniera non dissimile dalle altre, le differenti varianti di SRAM testate.

Tabella : Lista delle varie configurazioni di SRAM testate

Nome Simbolico	*Entrate*	*Banchi*
SRAM_Cache_IndexTrick_8GB_1Bank	262144	1
SRAM_Cache_IndexTrick_8GB_2Bank	262144	2
SRAM_Cache_IndexTrick_8GB_4Bank	262144	4
SRAM_Cache_IndexTrick_16GB_1Bank	524888	1
SRAM_Cache_IndexTrick_16GB_2Bank	524888	2
SRAM_Cache_IndexTrick_16GB_4Bank	524888	4
SRAM_Cache_IndexTrick_32GB_1Bank	1048576	1
SRAM_Cache_IndexTrick_32GB_2Bank	1048576	2
SRAM_Cache_IndexTrick_32GB_4Bank	1048576	4

I risultati delle simulazioni sono riassunti nei grafici che seguono.

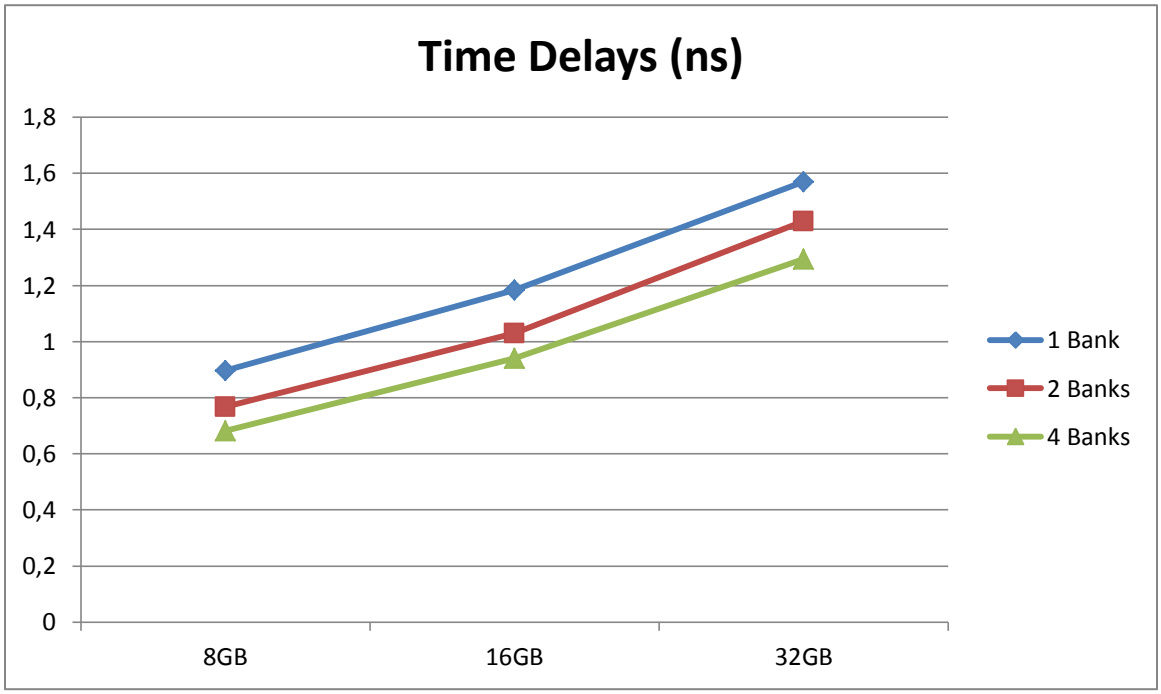

Grafico : Latenze per le varie SRAM

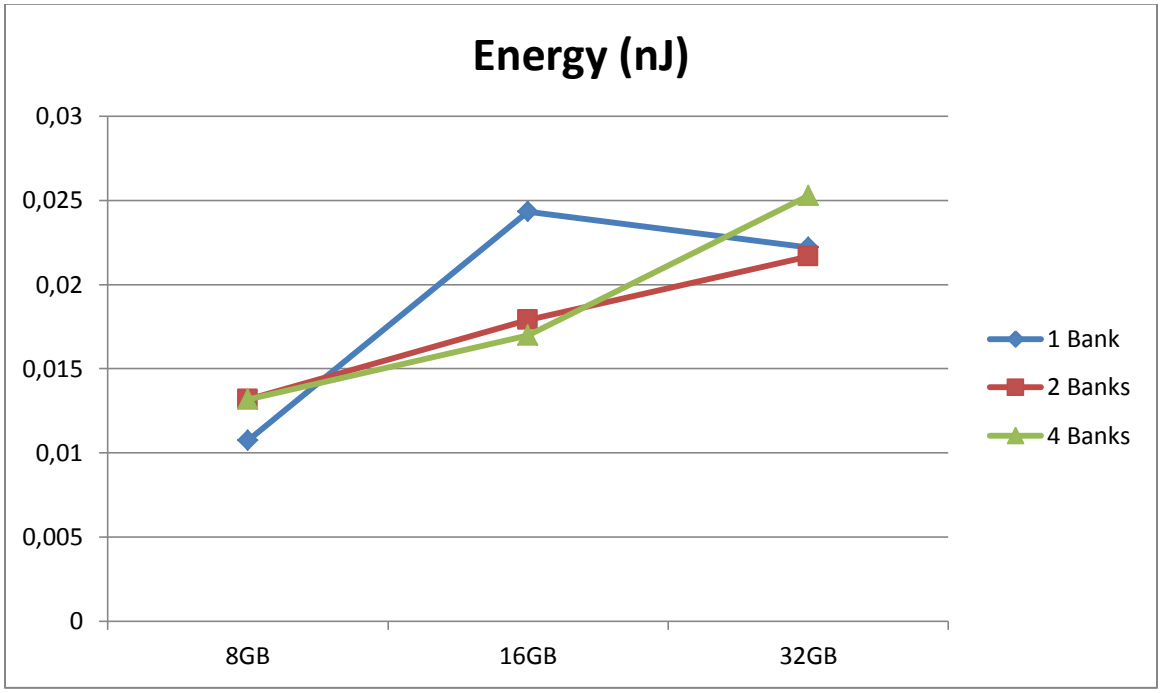

Grafico : Energia consumata da un singolo accesso nelle varie SRAM

Analizzando i grafici possiamo innanzitutto notare una differenza con le analisi precedenti: non è presente il grafico relativo all'andamento dell'area in funzione della

dimensione e del numero di banchi. Questa è una scelta voluta in quanto, come già più volte ricordato, la logica di controllo sarà posizionata fisicamente al *layer* 1 e quindi, essendo questo *layer* praticamente occupato solo da queste componenti e dai *row buffers*, risulta essere praticamente vuoto.

Passando invece ai *delay* notiamo come, aumentando la dimensione della SRAM, il *delay* aumenta mentre diminuisce, seppur di poco, se si aumenta il numero di banchi poiché aumenta il parallelismo.

L'energia invece possiede uno strano andamento. Le linee di referenza non mostrano più un andamento parallelo e regolare ma hanno un comportamento anomalo: questo ci permette dunque di discernere le diverse scelte progettuali. Si è scelto la versione a 1 banco per la SRAM indirizzante 8 GB di DRAM (il *delay* è maggiore ma comunque inferiore ad 1 ciclo di clock a $3GHz$ mentre l'energia è minore), 4 banchi per la versione indirizzante 16 GB di DRAM (miglior *delay* e miglior energia) e infine la versione a 2 banchi per la SRAM da affiancare a 32 GB di memoria.

3D-DRAM

È ora il momento di analizzare forse la principale componente del sistema CMM: la vera e propria 3D-DRAM. Come già indicato più volte l'insieme di dimensioni adottate per le soluzioni hardware è pari a 8, 16 e 32 GB. Anche in questo caso tuttavia, come è accaduto per la SRAM, è possibile effettuare delle suddivisioni in base al numero di canali, di ranchi e banchi.

Il numero di canali (*channels*) è stato fissato ad 1. Questa scelta, sebbene penalizzi molto il parallelismo, da un lato è stata effettuata in quanto vere e proprie realizzazioni pratiche di memorie nella versione True-3D non sono ancora state prodotte e, dall'altro lato, è stata effettuata per poter comparare una vera e propria 3D-Memory Entry Level con una baseline da tempo affermata ed efficiente.

Il numero di ranghi, seguendo il lavoro di (8) e (9), è invece stato fissato a 4. Questo garantisce una buona simmetria dei singoli *layer* e non inficia molto lo spazio e la densità di TSV da interconnettere. Da non sottovalutare è infatti la limitazione che il numero di ranghi deve essere potenza di 2.

Il numero di banchi è invece stato mantenuto variabile e, fissato il numero dei canali e il numero di ranchi, sono state effettuate diverse simulazioni. La tabella seguente mostra tutte le varianti analizzate.

Tabella : Varianti della DRAM simulate in CACTI

Nome Simbolico	*Grandezza (GB)*	*Banchi*	*Dies*
3D-DRAM_8GB_8Banks_32KBpP_4Dies	8	8	4
3D-DRAM_8GB_8Banks_32KBpP_8Dies	8	8	8
3D-DRAM_8GB_16Banks_32KBpP_4Dies	8	16	4
3D-DRAM_8GB_16Banks_32KBpP_8Dies	8	16	8
3D-DRAM_16GB_8Banks_32KBpP_4Dies	16	8	4
3D-DRAM_16GB_8Banks_32KBpP_8Dies	16	8	8
3D-DRAM_16GB_16Banks_32KBpP_4Dies	16	16	4
3D-DRAM_16GB_16Banks_32KBpP_8Dies	16	16	8
3D-DRAM_16GB_32Banks_32KBpP_4Dies	16	32	4
3D-DRAM_16GB_32Banks_32KBpP_8Dies	16	32	8
3D-DRAM_32GB_16Banks_32KBpP_4Dies	32	16	4
3D-DRAM_32GB_16Banks_32KBpP_8Dies	32	16	8
3D-DRAM_32GB_32Banks_32KBpP_4Dies	32	32	4
3D-DRAM_32GB_32Banks_32KBpP_8Dies	32	32	8
3D-DRAM_32GB_64Banks_32KBpP_4Dies	32	64	4
3D-DRAM_32GB_64Banks_32KBpP_8Dies	32	64	8

I grafici seguenti mostrano i risultati delle simulazioni. Si ricordi che, fissata la dimensione della DRAM, scopo di queste simulazioni è quello di determinare la coppia $\langle \#banks, \#dies \rangle$.

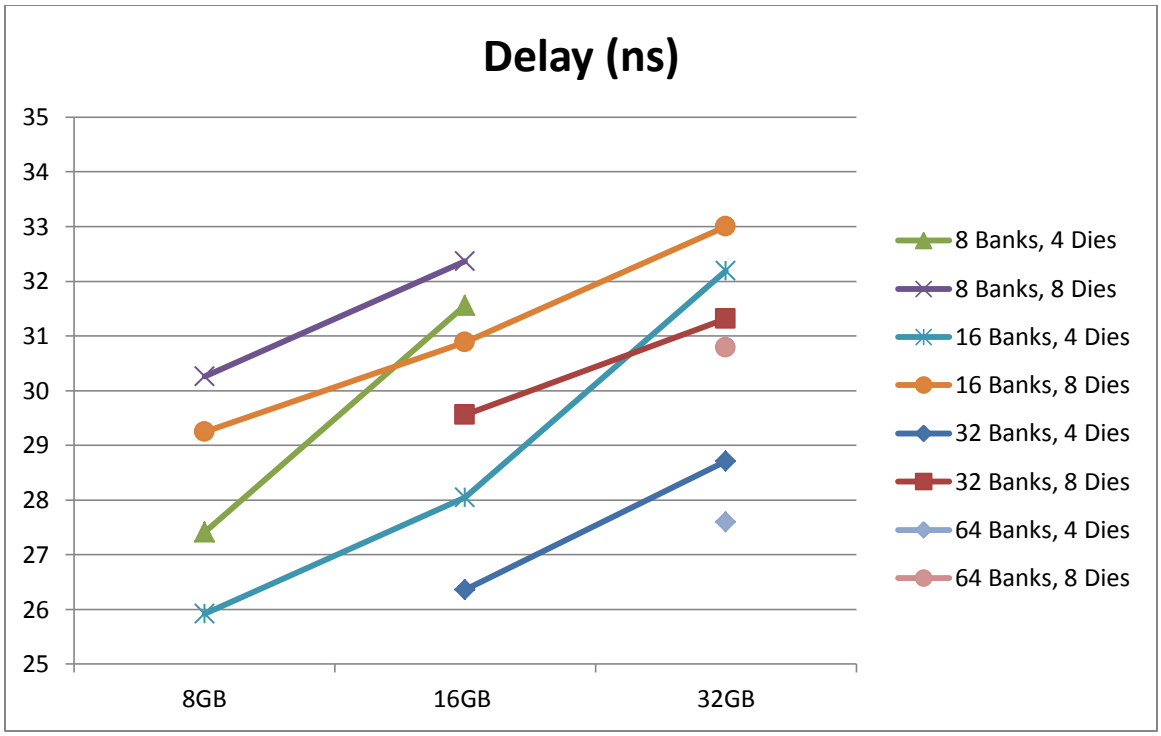

Grafico : Latenze di accesso medie per ogni tipo di memoria considerate

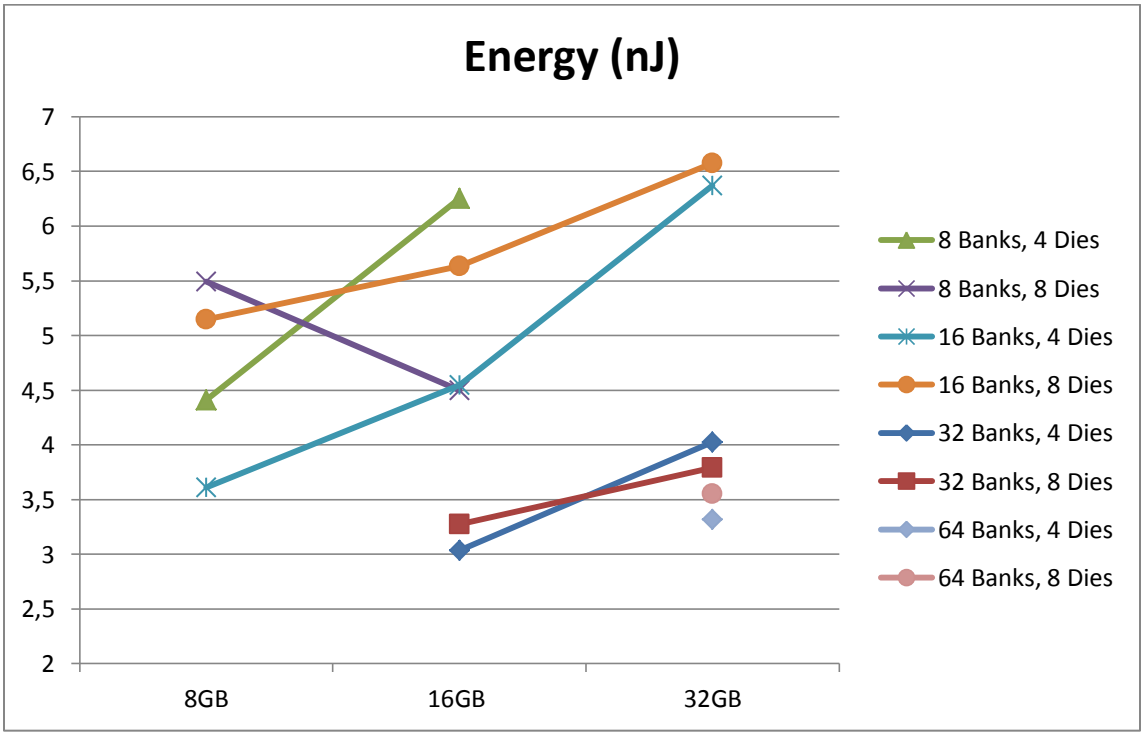

Grafico : Energia consumata durante un accesso della durata media in DRAM

Una prima analisi della Tabella e dei grafici mostra come non tutte le combinazioni siano state simulate. Questo ha la sua motivazione nel fatto che si è ritenuto opportuno non avere un banco con dimensione superiore a 512 MB.

Una prima possibile cernita riguarda l'eliminazione dalle scelte possibili di tutte le alternative che coinvolgono 8 differenti *dies*. Tali alternative infatti hanno tempi di risposta ed energia consumata sempre superiori rispetto alle loro corrispettive versioni a 4 dies.

Tra le versioni a 4 *dies* dunque, partendo dunque dalla memoria avente dimensione 8 GB si nota come sia per le latenze che per l'energia consumata l'alternativa migliore risulta essere quella composta da 16 banchi. Un discorso analogo è fattibile considerando le coppie 16 GB, 32 banchi e 32 GB, 64 banchi.

Questo porta ad avere una sorta di regola empirica per quanto riguarda le DRAM: l'alternativa migliore tende ad essere quella che possiede un numero di banchi doppio rispetto alla dimensione in GB della memoria, quanto meno nel *range* [8,32] GB.

Valori ottenuti da altri Studi

Altro componente importante del sistema CMM nonché di tutte le varianti ad esso collegate è la PCM, sia essa usata come rimpiazzo per lo *storage* secondario sia esso utilizzato come parte integrante della memoria principale.

Una variante di CACTI, denominata NVSim, è stata sviluppata appositamente per la simulazione delle memorie PCM tuttavia poiché tale simulatore è ancora nelle fasi iniziali dello sviluppo e soprattutto poiché esso simula solo fino al livello di banco e non di modulo, non è stato utilizzato per le simulazioni del componente PCM.

Al fine però di ottenere dati realistici per la modellazione delle PCM il lavoro effettuato è stato quello di rielaborare gli studi di (12).

Quereshi infatti suppone che una memoria PCM sia 4 volte più lenta e 4 volte più densa di una DRAM planare. Data dunque una memoria classica (bidimensionale planare e *off-chip*) da n GB la corrispondente memoria PCM sarà composta da $4n$ GB. Ricordando quanto già esplicato precedentemente nei paragrafi introduttivi sulla tecnologia PCM, ogni singola cella di materiale calcogeno può memorizzare diverse combinazioni di bit semplicemente assumendo un valore di resistenza che consenta

differenti valori di corrente al suo interno. Poiché si presuppone una densità 4 volte superiore a quella della DRAM di riferimento è semplice stabilire come ogni cella PCM contenga fino a 2 diversi bit.

Per quanto invece riguarda la velocità di accesso un problema molto sentito nelle PCM è la differenza di velocità tra le operazioni di lettura e quelle di scrittura. Le operazioni di scrittura infatti prevedono diverse alternative nel caso debba essere scritto un bit 1 o un bit 0 e tali alternative richiedono tempi diversi. (Figura : Cella elementare di memoria e tempi delle operazioni). I tempi di accesso reali simulati quindi risultano come segue. Un file sorgente completo per la configurazione dei delay della PCM è riportato in APPENDICE B: File Sorgenti.

```
/// Time for a read (static)
#define t_READ 99.3243375
/// Time for a write (static). Suppose a ¾ slow operation_ and a ¼ fast operation.
#define t_WRITE t_READ * 8
```

Anche per quanto riguarda l'energia, partendo da una memoria 4 volte più lenta ma soprattutto 4 volte più densa, non è del tutto sbagliato imporre 4 come fattore di scala tra i valori dell'energia consumata dalla DRAM di base e quella consumata dalla PCM.

Gli studi effettuati da Quereshi indicano tuttavia che sfruttando la durabilità dei dati nella memoria non devono essere effettuate operazioni di *refresh*. Inoltre, poiché la lettura non è distruttiva anche in questo caso si può risparmiare l'energia che nelle DRAM era utilizzata per la riscrittura nella *bitline*. Infine è possibile risparmiare anche sulla scrittura semplicemente riscrivendo nella *bitline* solo la porzione di *row buffer* effettivamente modificata.

Tutti gli accorgimenti appena esposti, come è possibile notare dall'esame della Tabella , consentono di ridurre il fattore di scala da 4 a 1.3.

Tabella : Valori di progetto ottenuti per la PCM

Nome Parametro	*Valore in DRAM*	*Valore in PCM*	*Fattore di Scala*
Size (GB)	2	8	4
t_READ (ns)	24.831084375	99.3243375	4
t_WRITE (ns)	24.831084375	794.5947	32

READ_ENERGY (nJ)	0.862207	1.1208691	1.3
WRITE_ENERGY (nJ)	0.862209	1.1208717	1.3

Tabelle simili sono ottenibili inserendo i valori delle DRAM a 4 e 8 GB ottenendo così i rispettivi parametri per PCM dalla dimensione di 16 e 32 GB.

Simulazioni e analisi dei risultati ottenuti con SIMICS

Come già detto nei relativi capitoli, le nostre simulazioni sono state effettuate attraverso Simics, un simulatore *full-system*. Simics contiene al suo interno un modulo modificabile dall'utente, denominato `g-cache` il quale consente di tenere traccia dei comportamenti di una gerarchia di memorie cache e che permette di valutare i ritardi introdotti dalle operazioni di memoria. `g-cache`, già rielaborato nell'ambito di (13), è stato pesantemente modificato al fine di ottenere risultati molto accurati in termini di *delay*, per quanto riguarda i ritardi introdotti da componenti quali 3D DRAM, TLB, SRAM e, nel caso in cui questa fosse usata come memoria principale, PCM, nonché per la valutazione del consumo di potenza. La macchina su cui sono state effettuate le simulazione è un processore $x86-64$ "Hammer" a 4 *core* e ne è stata supposta una frequenza operativa di 3GHz. Il sistema operativo usato è Ubuntu 8.04.

Per valutare le diverse architetture sono state utilizzate due differenti suite di *benchmark*:

1. SPEC2006 *Benchmark* per cui sono stati utilizzati diversi mix al fine di ottenere differenti carichi di memoria.
2. OLTP *Cloud Benchmark*: essi sono una nuova categoria di benchmark i quali simulano server appartenenti a diverse categorie di servizi web. Essi infatti simulano non solo il server in sé, ma creano anche un carico di lavoro personalizzabile costituito da svariati client, in modo da ottenere differenti condizioni operative e carichi di lavoro.

I *mix* utilizzati nell'ambito della suite SPEC2006 sono i medesimi utilizzati in (13). Per quanto riguarda gli OLTP, la seguente tabella illustrerà i diversi mix utilizzati. Da notare come il *footprint* di ciascun mix è nell'ordine dei 15 GB, anche se, facendo

questi *mix* capo ad un server MySQL, raggiungono un *footprint* dell'ordine dei 20 GB. Ciascun *benchmark* è stato assegnato ad un solo core al fine di poter tracciare con precisione il numero di istruzioni eseguite. I *benchmark* **sjeng** e **stream** sono stati utilizzati al fine di stressare di più il sistema e simulare le operazioni aggiuntive che un server esegue e che non riguardano direttamente l'attività di gestione di richieste remote.

Tabella : Benchmark mix per quanto riguarda i cloud benchmark.

Nome Mix	*Benchmark 1*	*Benchmark 2*	*Benchmark 3*	*Benchmark 4*
Auctionmark	Auctionmark	Auctionmark	Stream	Sjeng
Epinions	Epinions	Epinions	Stream	Sjeng
Tatp	Tatp	Tatp	Stream	Sjeng
Seats	Seats	Seats	Stream	Sjeng

Ciascuna simulazione ha eseguito inizialmente 500 milioni di istruzioni al fine di inizializzare al meglio le varie strutture di g-cache. Successivamente ogni simulazione ha eseguito istruzioni finché uno dei benchmark non avesse raggiunto un miliardo di istruzioni eseguite. Per quanto riguarda invece il consumo di potenza ciascuna simulazione ha preso in considerazione solamente la potenza dinamica assorbita dai vari componenti in oggetto a questo studio. In particolar modo sono state prese in considerazione la potenza dinamica della 3D DRAM, delle varie SRAM, dei TLB e, nel caso di utilizzo di PCM come memoria principale, anche della PCM stessa.

CMM: Simulazioni e analisi dei risultati

Il lavoro di perfezionamento effettuato sulla CMM è disponibile in (2).

3D-DRAM come LLC: Simulazioni e analisi dei risultati

Ricordando lo schema semplificativo riguardante la 3D-DRAM utilizzata come last level cache e riportato precedentemente in Figura si può innanzitutto notare che i componenti effettivamente simulati sono due.

Inoltre un'altra decisione progettuale che è stata effettuata e che vale la pena menzionare è stata quella di considerare la *last level cache* con una grandezza di diversi ordini di grandezza più grande rispetto a quelle attualmente poste nei processori attuali.

Infatti l'ultima serie di processori Intel per il mercato utente, la serie **i7 Ivy Bridge**, è formata da una cache di ultimo livello condivisa dalle dimensioni che non superano i 15 MB.

Accoppiare una cache di ultimo livello tanto piccola ad una PCM utilizzata come memoria principale tuttavia non risulta essere una buona scelta poiché, come noto, le PCM non supportano un gran numero di scritture nella stessa locazione. Questo è dunque il motivo principale che ha portato alla scelta di 3D-DRAM di grandezza notevole e significativa.

Ricordando inoltre gli studi di (12) si è scelto di mantenere, per quanto possibile, un rapporto tra grandezza della 3D-DRAM e quello della PCM che fosse nell'ordine di 1/4.

La tabella seguente mostra tutte le combinazioni di grandezza di memoria 3D-DRAM che sono state considerate per le simulazioni che verranno riportate nel seguito.

Tabella : Lista delle configurazioni di 3D-DRAM usata come LLC simulate

Nome Simbolico	*Grandezza (GB) 3D-DRAM*	*Grandezza (GB) PCM*
3D_LLC_2G_PCM_8G	2	8
3D_LLC_4G_PCM_16G	4	16
3D_LLC_2G_PCM_32G	2	32
3D_LLC_4G_PCM_32G	4	32
3D_LLC_8G_PCM_32G	8	32

Un'ulteriore scelta che è stata effettuata è stata quella di considerare accuratamente le suite di *benchmark* utilizzate. Infatti i *benchmark* appartenenti alla suite SPEC2006 sono dei *benchmark* relativamente esaustivi ma che tuttavia hanno un'occupazione di memoria relativamente blanda, in particolare per i *mix* utilizzati da (13) e riutilizzati in questo studio, come già esposto nell'introduzione di questa sezione, la memoria utilizzata di picco non supera mai i 2.2 GB. Questa limitata occupazione di memoria, unita alle grandi dimensioni che sono state date sia alla LLC che alla PCM portano in maniera naturale alla sola esecuzione di tali *benchmark* per la combinazione di 2 GB di *last level cache* e 8 GB di PCM come memoria principale.

Per il test delle altre configurazioni invece si è optato per l'utilizzo dei *benchmark* OLTP. Tali *benchmark* infatti come ampiamente discusso in precedenza sono riconfigurabili nel loro carico di lavoro e pertanto si è riusciti ad ottenere in maniera relativamente elementare delle suite di *benchmark* il cui *working set* privato di picco raggiungesse i 25 GB con un *working set* privato medio di circa 16 GB. Le suite di *benchmark* OLTP utilizzate sono state precedentemente riportate in Tabella .

Espansione di Simics per l'aggiunta dei moduli LLC e PCM-MM

Ogni modulo all'interno di Simics è scritto in linguaggio `C` o `C++.` Qualunque sia il linguaggio scelto dal progettista tuttavia ogni modulo è interpretabile, ad alto livello ed utilizzando un linguaggio proprio della programmazione ad oggetti, come una classe che implementa due particolari interfacce. Di seguito sono riportati in pseudo-codice le dichiarazioni di tali interfacce.

```
public interface class_data_t
{
    public delegate void new_istance();
    public string description;
}


public interface timing_model_interface_t
{
    public delegate cycle_t operate(conf_object_t* thisObject,
                                    conf_object_t* memorySpace,
                                    map_list_t* memoryMap,
                                    generic_transaction_t* memoryOperation);
}
```

La prima delle due interfacce associa, all'oggetto Simics generico, un costruttore (il puntatore a funzione delegato `new_instance`) e una descrizione sotto forma di stringa.

La seconda interfaccia invece associa all'oggetto il metodo `operate` ossia l'*entry point* da chiamare quando si verifica un'operazione di memoria che coinvolge questo componente. I parametri di questa funzione sono:

- **`thisObject`**: puntatore alla classe base `conf_object_t` da cui tutti gli oggetti Simics derivano. Tramite questo oggetto, de-referenziandolo al tipo di oggetto istanziato nel modulo corrente, è possibile accedere ai campi dell'oggetto di memoria corrente in maniera da tracciarne statistiche.

- **`memorySpace`**: Non utilizzato
- **`memoryMap`**: Non utilizzato
- **`memoryOperation`**: L'operazione di memoria per la quale è stato chiamato questo modulo.

Nello specifico, al fine di simulare un modulo di cache, qualunque siano le sue caratteristiche, all'interno di Simics è opportuno e risparmioso in termini di tempo partire come base dal modulo g-cache. Tale modulo infatti implementa in maniera nativa:

- Connessione alle interfacce precedentemente citate
- Connessione con le cache di livello inferiore
- Connessione con le cache di livello superiore
- Protocollo di coerenza in caso di cache condivisa
- *Tagging* Fisico o Virtuale
- Indirizzamento Fisico o Virtuale

Ovviamente alcuni dei punti dell'elenco precedente andranno modificati a seconda di quale livello di cache il modulo andrà ad implementare. Si consideri a titolo di esempio la cache di primo livello, sia essa *instruction cache* o *data cache*. In tal caso non esiste alcuna cache di livello inferiore ma solo l'*instruction splitter* che indirizzerà le transazioni di memoria verso la cache di livello 1 dedicata alle istruzioni in caso di *instruction fetch* e verso la cache di livello 1 dedicata ai dato in caso di istruzione con operandi in memoria.

Nel caso della LLC tuttavia non si verifica questo tipo di mancanza bensì si verifica il caso opposto: la cache di livello superiore non esiste e pertanto si è collegato il modulo g-cache direttamente al modulo implementante la memoria principale.

La tabella seguente mostra come siano collegati internamente i vari moduli e se tali moduli siano nativi di Simics, modificati a partire da moduli pre-esistenti o totalmente custom.

Tabella : Lista dei moduli caricati all'interno di Simics per le simulazioni

Nome Modulo	*Timing Model Chiamante*	*Timing Model Chiamato*	*Tipo di Modulo*	*Modulo Originale*
id-splitter	N/A	l1_i_cache_x l1_d_cache_x	Nativo	id-splitter
l1_i_cache_x l1_d_cache_x	id-splitter	l2_cache_x l2_cache_x	Modificato	g-cache
l2_cache_x	l1_i_cache_x l1_d_cache_x	llcache	Modificato	g-cache
llcache	l2_cache_x	pcm-mm	Modificato	g-cache
pcm-mm	llcache	frtd	Modificato	Cmm
frtd	pcm-mm	N/A	Custom	N/A

Partendo dall'*id-splitter* si nota, come già accennato, che esso è capace di discernere tra istruzioni che richiedono istruzioni (istruzioni di *fetch*) e istruzioni che richiedono dati. Effettuata la distinzione quello che avviene è che verrà chiamato il modulo opportuno tra l1_i_cache_x e l1_d_cache_x. Di questi moduli esistono in generale N diverse istanze, una per ogni core da cui il processore è formato.

Analogamente, uno tra i due moduli l1_i_cache_x e l1_d_cache_x avrà il compito di chiamare il prossimo *timing model* della gerarchia e così via.

Prima di passare all'esplicazione dei due principali moduli richiesti da questo studio, ossia **llcache** e **pcm-mm**, è necessario prima comprendere il concetto di *timing model*.

Il timing model

Ricordando gli stralci di pseudocodice riportati precedentemente, il funzionamento generale di Simics è, a grandi linee, molto semplice. Esso non è infatti molto diverso da una macchina virtuale. Le istruzioni tuttavia non sono eseguite direttamente sul *core* fisico (eventualmente posto in uno stato di protezione intermedio come accada in sistemi di virtualizzazione attuali quali Virtual PC, VirtualBox eccetera) ma bensì su un processore software simulato che quindi porta Simics ad essere più vicino ad un interprete/compilatore JIT come lo può essere la Java Virtual Machine o CLR.

Al termine di ogni istruzione, se tale istruzione referenziava la memoria, Simics confeziona un oggetto di tipo `generic_transaction_t` nel quale inserisce tutte le informazioni sulla transazione quale indirizzo virtuale, indirizzo fisico, core di provenienza eccetera e, verificata la presenza di un modulo capace di accettare richieste di memoria, lo passa a tale modulo. A questo punto il processore virtuale di Simics è fermo e verrà eseguita tutta la gerarchia di chiamata data dalla composizione dei moduli.

Ogni modulo è caratterizzato dalla presenza di un *entry point* denominato `operate`. Tale *entry point* ha come tipo di ritorno il tipo interno di Simics `cycles_t`, alias per `uint16_t`. Al termine della gerarchia di chiamata quello che teoricamente un buon programmatore dovrebbe fare, come effettivamente è stato realizzato in questo studio, è quello di ritornare tramite un'istruzione `return` all'ambiente Simics il numero totale di cicli di clock che il sistema di memorie implementato dalla gerarchia impiegherebbe se effettivamente realizzato. Simics avrà dunque il compito di sommare tali cicli di clock al suo clock virtuale interno e questa operazione permetterà, a simulazione conclusa, di ottenere quanti cicli di clock ha impiegato, tra *stall* ed esecuzione, una certa suite di benchmark che ha eseguito per un determinato numero di istruzioni che, in tutti i risultati qui presentati, è stato posto uguale a 500,000,000 per l'operazione di *warming*, necessaria al fine di non falsare i risultati con troppe *cold misses*, e a 1,000,000,000 per le operazioni effettivamente tenute sotto osservazioni.

Il modulo llcache

Realizzato a partire da un modulo g-cache vergine, lo scopo del modulo **llcache** è quello di realizzare una **last level cache** che risponda all'ambiente Simics, non come una normale cache progettata e realizzata tramite tecnologia CMOS, bensì come una cache realizzata tramite l'unione di due separate componenti: una realizzata tramite logica statica, CMOS, denominata SRAM e l'altra, ottenuta tramite logica dinamica, che implementa la vera e propria memoria 3D-DRAM. Da questo punto di vista in effetti il modulo **llcache** non si presenta in maniera molto dissimile dal modulo CMM realizzato da (13) ed espanso da (2). La differenza risiede nel fatto che in questo caso il modulo **llcache** non deve implementare tutte le politiche necessarie allo smistamento di pagine e sottopagine in quanto esso deve comportarsi non come un ibrido tra cache e memoria

principale bensì come semplice cache semplicemente realizzata tramite una tecnologia differente da quella solita.

Programmare questo modulo quindi ha richiesto un intervento all'interno del modulo g-cache originale in maniera da integrarne il simulatore di *row buffer* per il calcolo delle *hit*, *semi-hit*, *semi-miss* e *miss* e in maniera da integrarne anche il corrispettivo modulo per il *profiling* dell'energia consumata. In APPENDICE B: File Sorgenti è riportato il codice per l'*header* del modulo aggiuntivo che è stato realizzato per la modellazione definitiva del modulo **llcache**. Di seguito è invece riportato il codice che realizza la simulazione dei row buffer all'interno della memoria 3D.

```
struct RowBuffer
{
	/// The base addresses related to a single address for a single
 rowbuffer related to a bank collection
	uint64_t BaseAddress[N_ROWBUFFER_PER_SET];

	/// The index of the head position in the list of BaseAddress
	int m_Head;

	/// Determines if this RowBuffer has been accessed at least once
	_Bool IsValid[N_ROWBUFFER_PER_SET];
};

/// Determines if the RowBuffers array has been initialized
_Bool initialized = FALSE;

/// The RowBuffers
RowBuffer RowBuffers[N_ROWBUFFERS_SET];
```

Il modulo pcm-mm

Il secondo modulo realizzato al fine di ottenere una precisa ed esauriente simulazione del sistema 3D-DRAM-LLC è quello relativo alla memoria principale vera e propria. Tale memoria, realizzata ovviamente con una tecnologia PCM, ha dunque la necessità di essere indirizzata e pertanto necessita di un meccanismo di traduzione degli indirizzi da virtuale a fisico.

Il primo modulo aggiuntivo che è stato realizzato dunque è stato quello del TLB, ente primario dedito alla veloce traduzione dell'indirizzo. A coadiuvare il TLB tuttavia è necessario avere un metodo di backup per la traduzione degli indirizzi qualora il TLB stesso produca una *miss*.

Ciò che accade nei sistemi attuali è, come noto dalla teoria dei sistemi operativi e dei calcolatori elettronici, l'accesso ad una serie di strutture residenti in memoria e delle quali il sistema conosce la locazione fisica e l'utilizzo di tali strutture per la traduzione dell'indirizzo. Risulta semplice concludere che il meccanismo appena citato altro non è che il *page walking*.

Al fine di evitare il *page walking* però si è cercato di utilizzare metodi già consolidati e utilizzati in (13) e in (2): l'indirizzamento *cache-like*. Nel caso della PCM-MM tuttavia tutta la logica dedicata alla traduzione dell'indirizzo non sarà posta direttamente al di sopra del processore bensì in un chip separato, eventualmente geograficamente vicino alle celle di PCM vere e proprie.

Risultati e analisi delle simulazioni

Si riportano qui di seguito i risultati ottenuti dalle simulazioni effettuate con Simics per l'architettura composta dalla 3D-DRAM come cache di ultimo livello e dalla PCM utilizzata quale memoria principale.

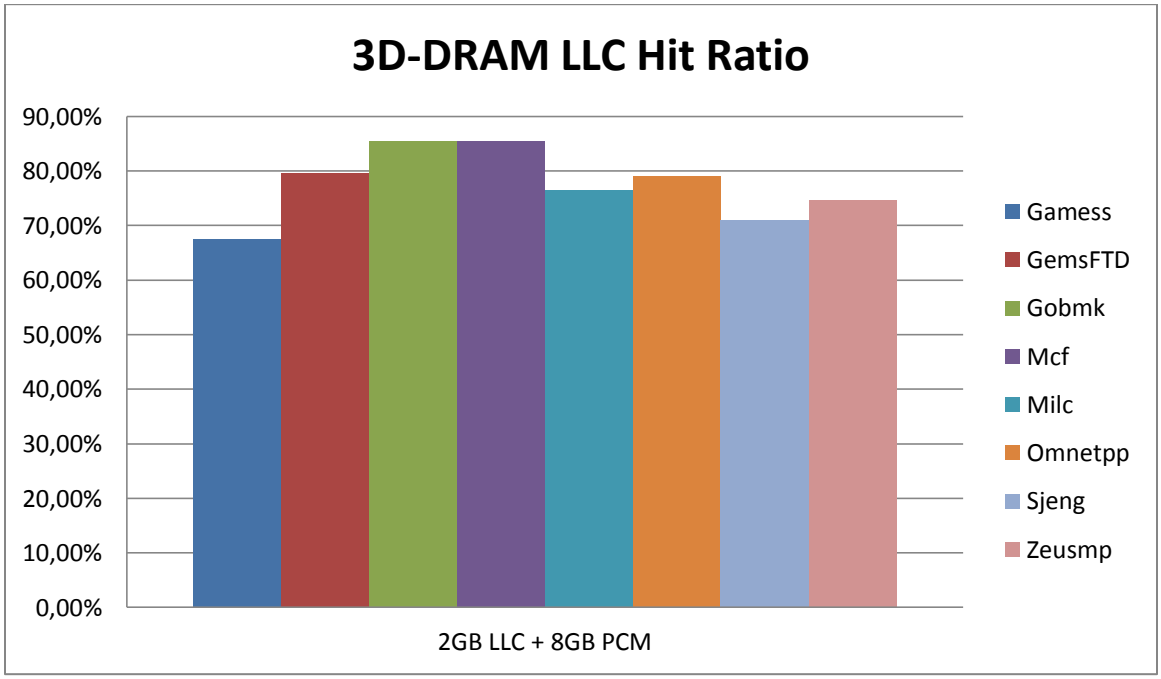

Grafico : Hit Rate della 3D-DRAM utilizzata come LLC (SPEC2006)

La bontà del modulo **g-cache** unita alle modifiche effettuate dall'autore hanno realizzato un modulo di cache capace di raggiungere fino al 85% di *hit rate*. La grandezza della 3D-DRAM considerata nel grafico precedente è di 2 GB e pertanto è quella relativa ai benchmark della suite SPEC2006.

Le altre configurazioni di LLC utilizzate invece in comune con i benchmark cloud OLTP hanno riportato i risultati esposti di seguito.

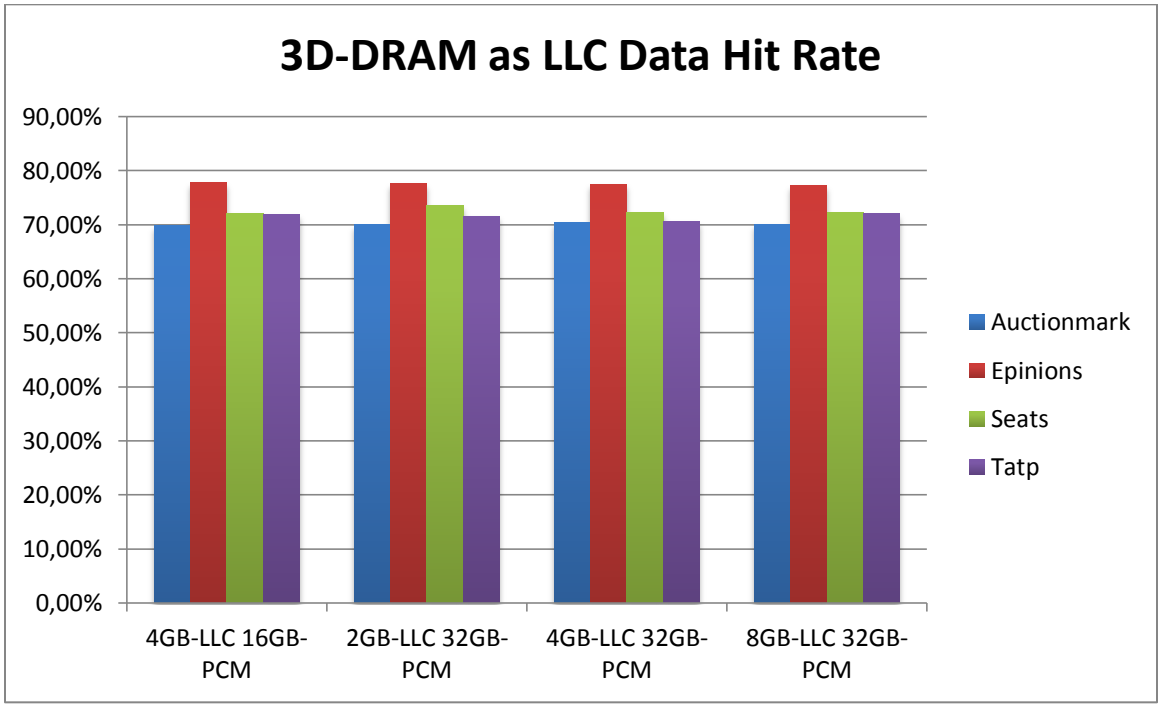

Grafico : Hit Rate della 3D-DRAM utilizzata come LLC (OLTP)

In questo caso, complice anche la più grande occupazione di memoria dei benchmark OLTP, si hanno risultati leggermente peggiori che tuttavia non inficiano significativamente la bontà del sistema essendo comunque valori superiori al 70%.

La *Hit Rate* media può essere calcolata semplicemente e vale:

$$\frac{1}{N_{benchmarks_{SPEC}}} \sum_{i=0}^{N_{benchmarks_{SPEC}}} HitRate[i] = 77.42\%$$

per i *benchmark* della suite SPEC2006 e

$$\frac{1}{N_{benchmarks_{OLTP}}} \sum_{i=0}^{N_{benchmarks_{OLTP}}} HitRate[i] = 72.90\%$$

per la suite OLTP.

Per quanto riguarda invece il numero di istruzioni per ciclo di clock il grafico seguente mostra la comparazione tra la tecnologia 3D-DRAM-LLC e la Baseline.

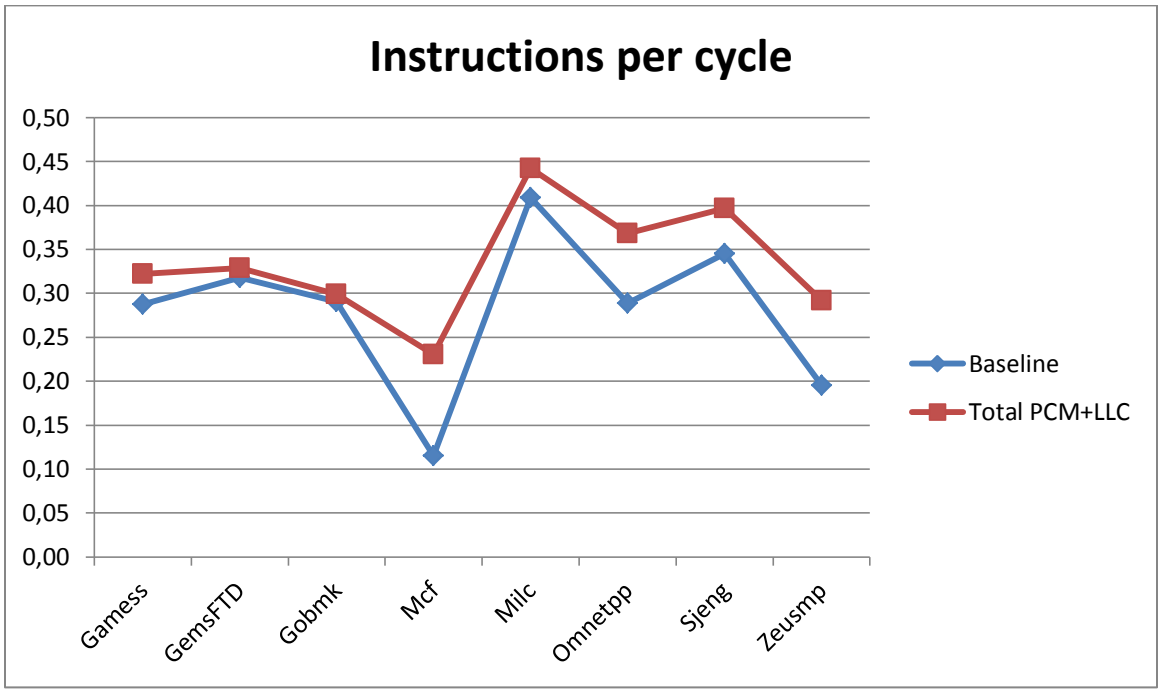

Grafico : Istruzioni per ciclo per l'architettura 3D-DRAM LLC (SPEC2006)

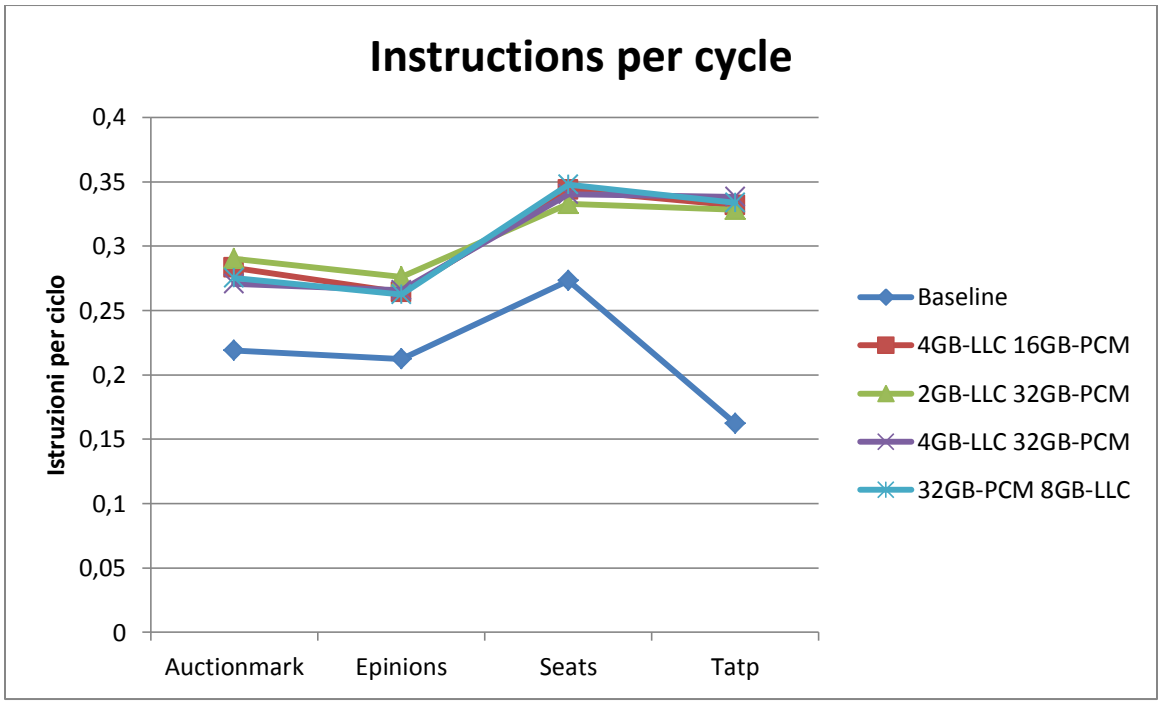

Grafico : Istruzioni per ciclo per l'architettura 3D-DRAM LLC (OLTP)

Il Grafico e il Grafico ben evidenziano la migliore esecuzione del sistema in esame tuttavia, al fine di garantire al lettore una più rapida comprensione dell'entità di tale miglioramento si considerino il Grafico e il Grafico

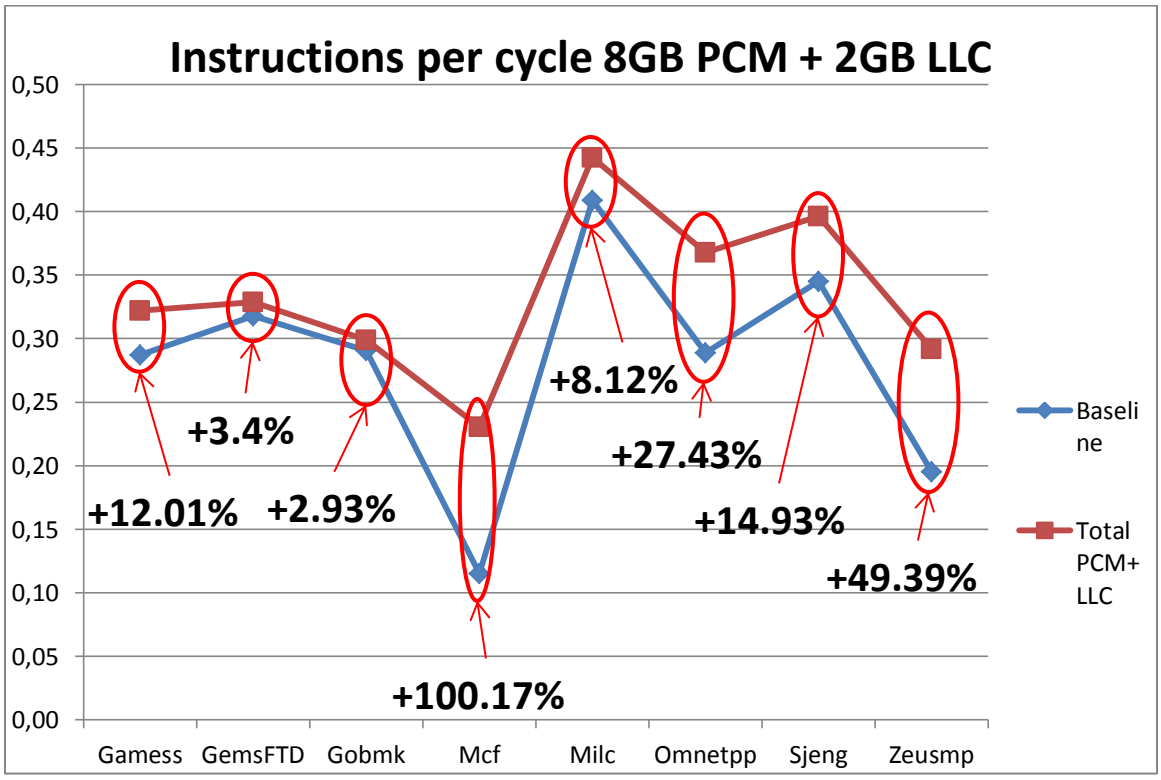

Grafico : Istruzioni per ciclo dettagliate, 3D-DRAM LLC - SPEC2006

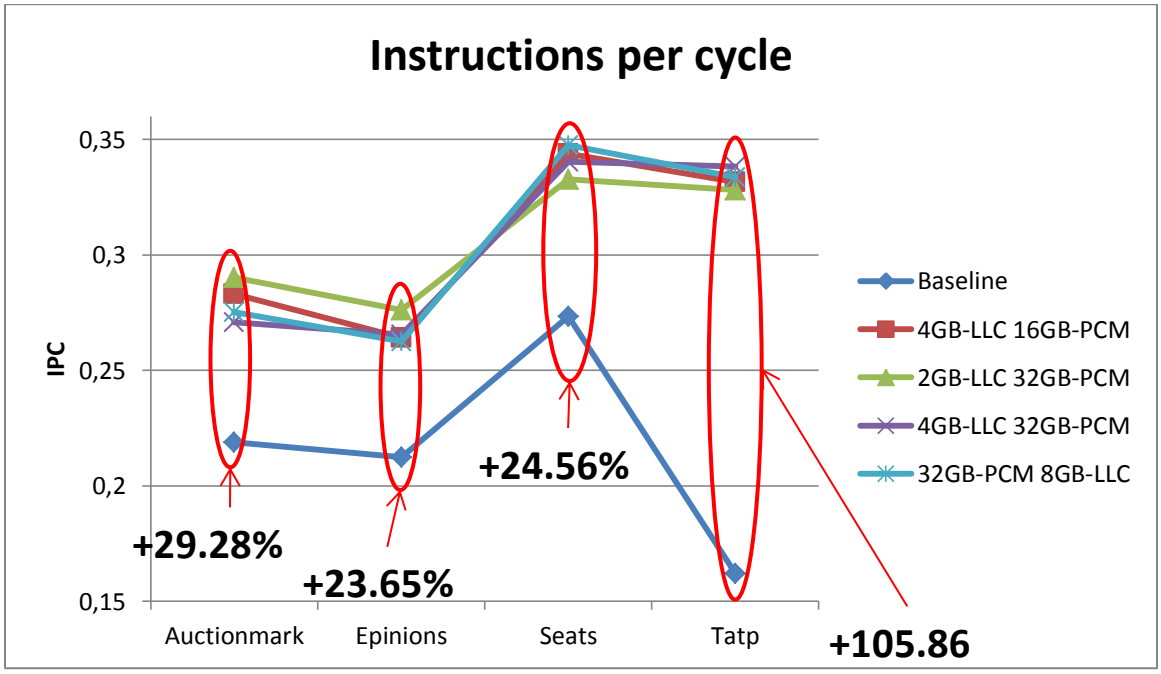

Grafico : Istruzioni per ciclo dettagliate, 3D-DRAM LLC – OLTP

I risultati sono molto positivi, specialmente per quanto riguarda due particolari *benchmark*, uno nella suite SPEC2006 e l'altro nella suite OLTP; tali benchmark sono **mcf** e **tatp**. Il diverso comportamento su questi *benchmark* è da reputarsi sicuramente causa delle diverse operazioni che il *benchmark* stesso effettua sulla memoria in quel lasso di operazioni comandate dal miliardo di istruzioni simulate.

L'andamento medio in ogni caso risulta positivo con un valore del 27.30% per la suite SPEC2006 e di 45% per la suite OLTP. Proprio per quest'ultima suite vi è da fare un'importante considerazione. Come si nota facilmente a partire dal Grafico e a seguire con il Grafico e il Grafico non vi è alcun beneficio visibile nell'avere una quantità di cache superiore ai 2 GB, non importa quale memoria PCM vi si affianchi. La scelta dunque da effettuare sembrerebbe essere quella di avere la combinazione 2 GB LLC + 32 GB di PCM-MM ed in effetti tale scelta è suggellata dall'andamento energetico riportato in: Grafico , Grafico , Grafico e Grafico .

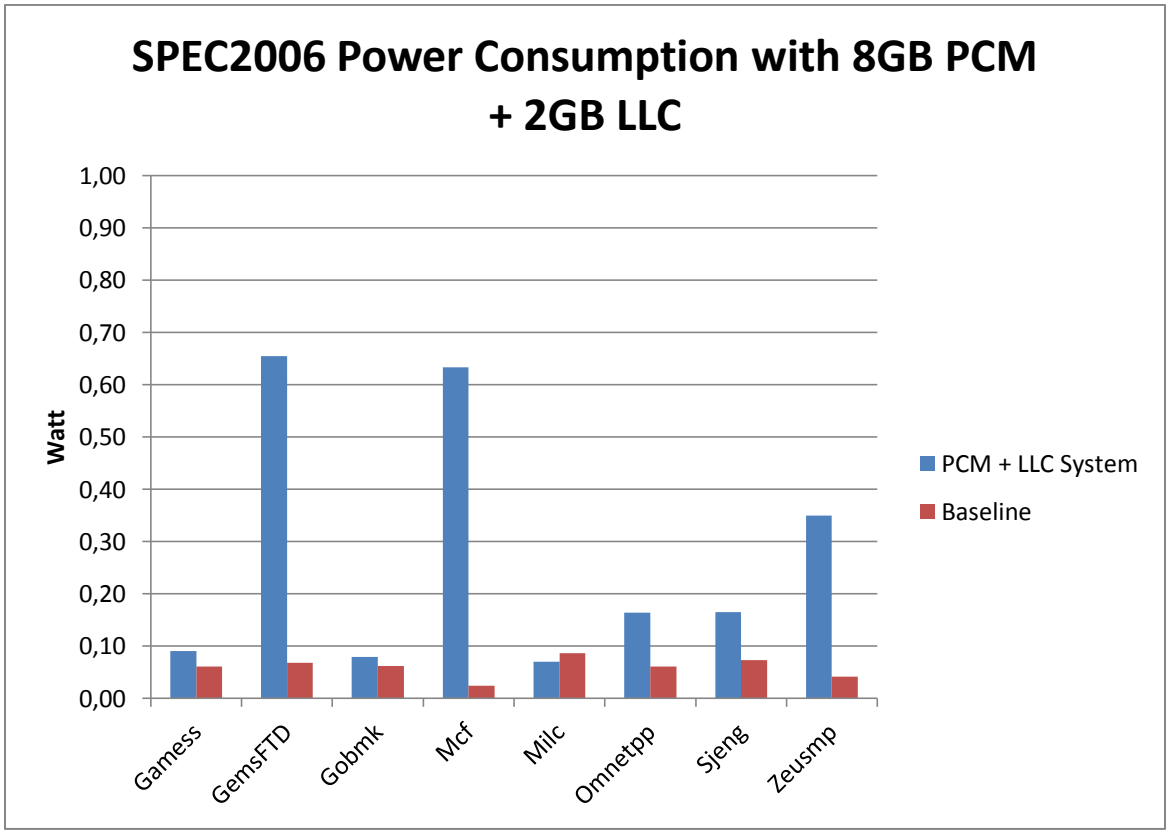

Grafico : Consumo energetico in Watt per la combinazione 2GB-LLC+8GB-PCM

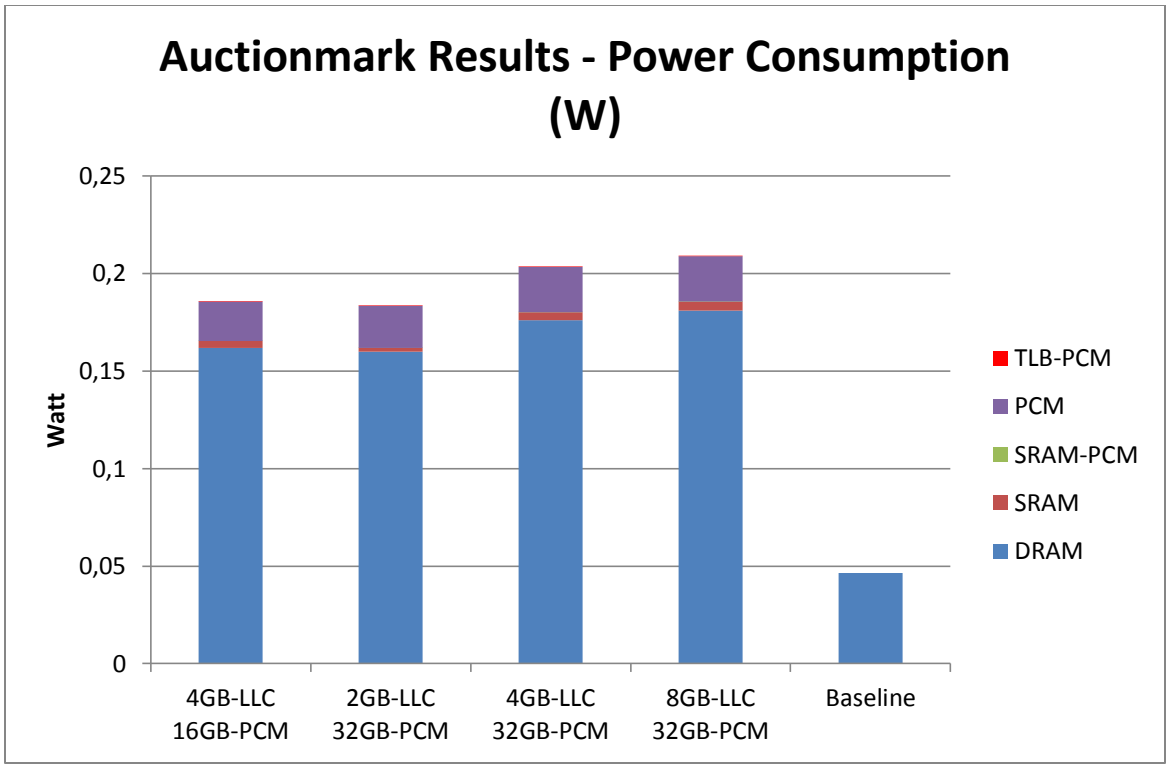

Grafico : Potenza assorbita del sistema 3D-LLC (Auctionmark)

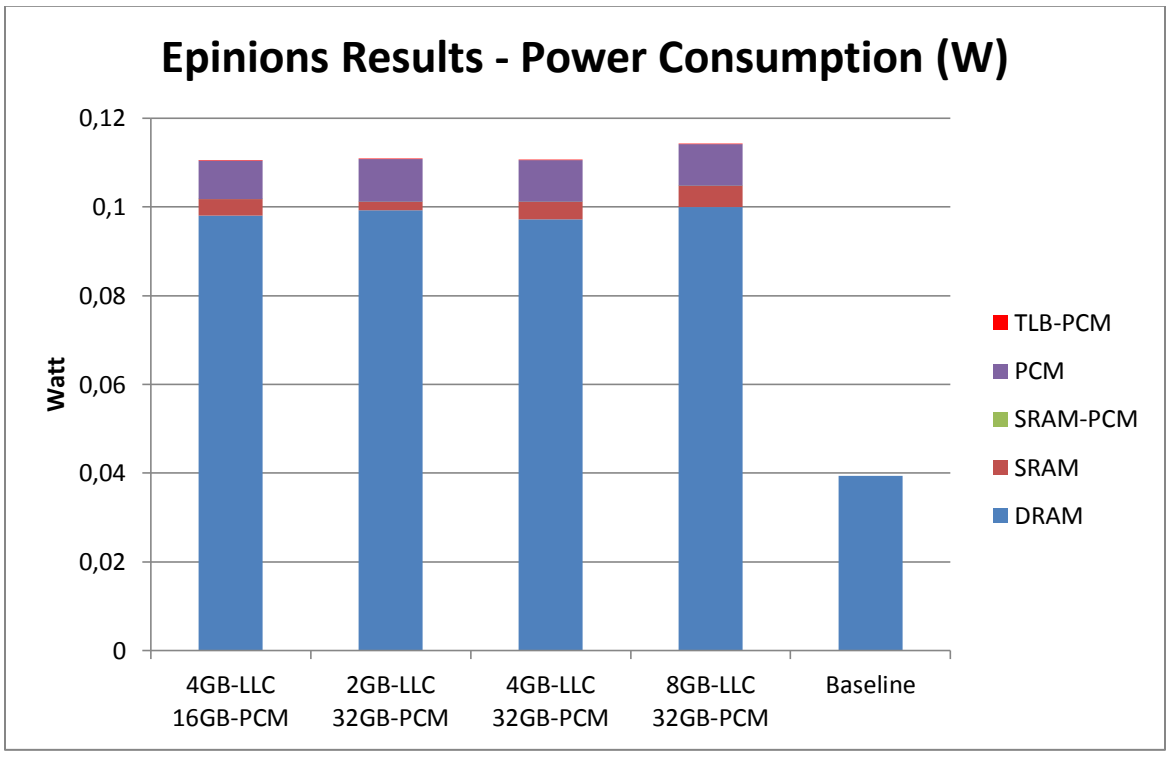

Grafico : Potenza assorbita dalle varie componenti del sistema 3D-LLC (Epinions)

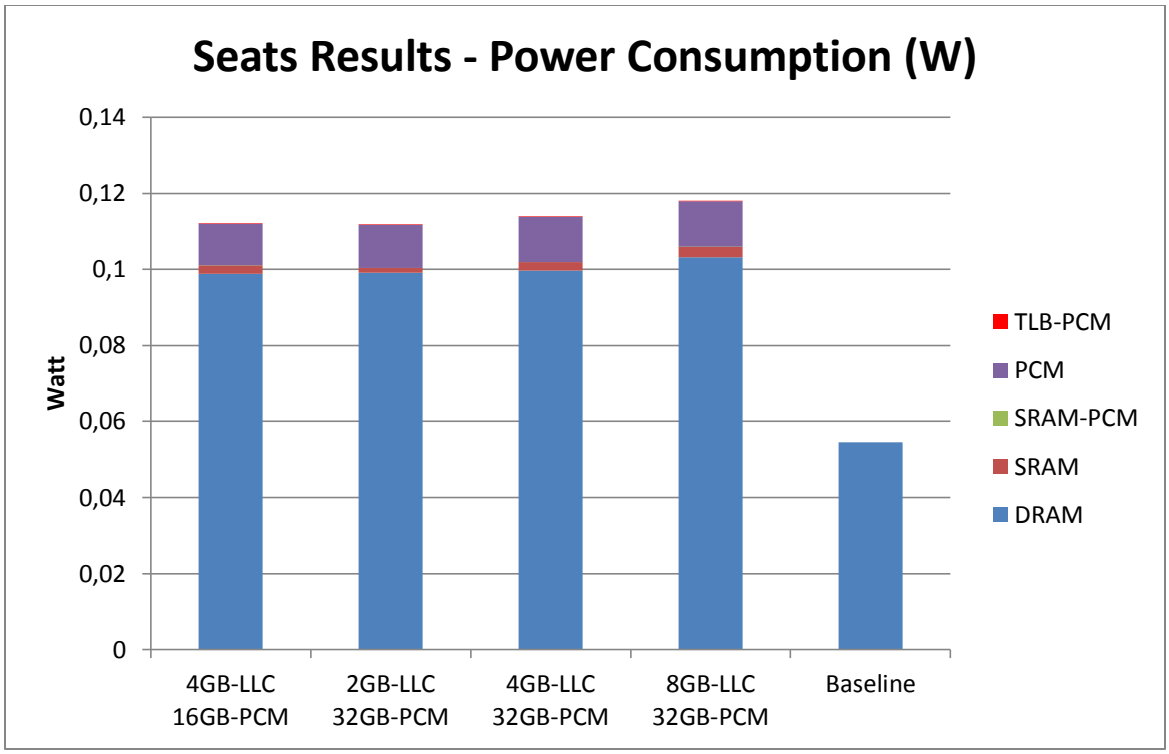

Grafico : Potenza assorbita dalle varie componenti del sistema 3D-LLC (Seats)

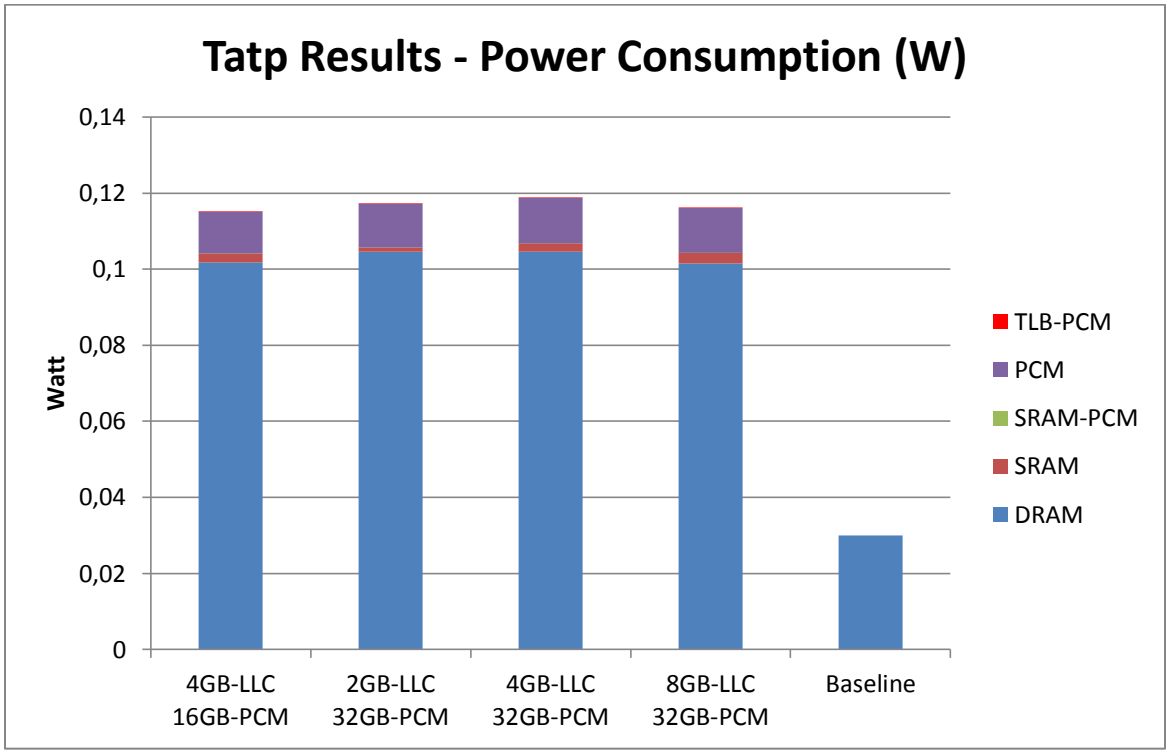

Grafico : Potenza assorbita dalle varie componenti del sistema 3D-LLC (Tatp)

Infatti si nota come la quantità totale di energia consumata è per la maggior parte dovuta alla componente dinamica (DRAM) e che questa componente risulta praticamente quasi indipendente dalla grandezza della DRAM stessa, sia essa 2 GB o 8 GB. Questo permette indubbiamente di poter scegliere, secondo le necessità e il bisogno, l'alternativa migliore possibile: ad esempio sarà possibile scegliere gran quantità di cache di ultimo livello per sistemi che necessitano di lavorare con grandi quantità di dati a intervalli brevi mentre si potrà scegliere alternative più risparmiose se si desidera avere un sistema più economico o che non necessita di istanziare diversi GB di memoria tutti insieme.

3D-DRAM come PMM: Simulazioni e analisi dei risultati

Ulteriore suddivisione e utilizzo della 3D-DRAM e della PCM può essere quello di considerare entrambe come parte della memoria principale. Lo studio approfondito di tale architettura è disponibile in (1).

CONCLUSIONI

Sono state analizzate diverse architetture di memoria, alcune realizzate ex-novo e altre rielaborate in seguito ad un precedente lavoro. Tutte le architetture sono alternative possibili per i sistemi ad alte prestazioni che, come ripetuto sin dall'introduzione, sono menomati dal problema del Memory Wall.

Dagli studi (1), (2) e ovviamente anche da questo, si nota come tutte le alternative risultano valide e ben superiori come performance rispetto alle opzioni attualmente presenti sul mercato.

Tutte le opzioni riportate inoltre presentano diversi margini di miglioramento che porteranno le simulazioni a dover essere rieseguite e i risultati a dover essere rianalizzati. I miglioramenti futuri che sono già stati programmati tuttavia sono miglioramenti il cui effetto sui risultati può solo essere positivo e, in tal caso, questo studio è uno studio che sottostima il sistema in esame e sovrastima (cfr. Baseline) l'architettura di riferimento.

Alcuni dei miglioramenti previsti coinvolgono l'introduzione dell'energia utilizzata per trasferire i vari dati e la migliore modellazione delle varie componenti utilizzando modelli matematici a minore approssimazione che garantiranno margini di errore molto inferiori a quelli attuali.

Per concludere con una metafora bellica, il *Memory Wall* è stato preso d'assalto e inizia a mostrare segni di cedimento. Gli esplosivi sono stati piazzati e ora tocca alle aziende produttrici di hardware, gli *artificieri*, innescare la miccia per abbattere le mura e conquistare una nuova terra inesplorata nell'ambito dell'architettura dei calcolatori.

RINGRAZIAMENTI

In questa sezione abbandono l'impersonale come scelta verbale e scrivo in prima persona: questa tesi non sarebbe stata scritta se non fosse stato per l'incoraggiamento che mi è stato dato durante tutto lo studio che vi è dietro: i miei genitori che mi hanno consentito di viaggiare fino in Texas per la stesura della tesi; i miei colleghi ma soprattutto amici che sono stato fortunato a trovare in un ambiente ostico come quello universitario, Stefano, Giandomenico e Domenico; il mio vicino di casa Sam che con la sua follia tipicamente americana mi ha fatto saggiare ciò che il Texas ha da offrire; gli amici lontani che anche con una sola parola sanno ribaltare i momenti un po' bui; infine tutte le persone che ho conosciuto qui in America che in un modo o nell'altro mi hanno fatto capire che esiste un mondo completamente nuovo, *a brand new world* citando una sigla televisiva, al di fuori della mia piccola ma amatissima penisola.

Una citazione importante la devo infine anche all'Università di Pisa e al suo staff di professori, assistenti e collaboratori che mi ha "ospitato" dal 2006 ad oggi.

Denton, Texas

3 maggio 2013

Giuseppe Regina

RIFERIMENTI

1. **Pianelli, Stefano.** *Memory Architectures Solutions using 3D-DRAM and PCM as Main Memory.* Pisa : Università di Pisa, 2013.

2. **Pisano, Giandomenico.** *Memory Architectures Solutions using 3D-DRAM.* Pisa : Università di Pisa, 2013.

3. *Cramming more components onto integrated circuit.* **Moore, Gordon.** 1965, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp, Vol. 1, pp. 1-4.

4. *Hitting the Memory Wall: Implications of the Obvious.* **Wulf, Wm. A. and McKee., Sally A.** 1994, University of Virginia, Vol. 1, pp. 1-3.

5. *Die Stacking (3D) Microarchitecture.* **Bryan Black, Murali Annavaram, Ned Brekelbaum John DeVale Lei Jiang Gabriel H. Loh Don McCauley Pat Morrow Donald W. Nelson Daniel Pantuso Paul Reed Jeff Rupley Sadasivan Shankar John Shen and Webb, Clair.** 2006. Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture.

6. **Utah, University of.** Wafer Bonding. [Online] 2006. http://www.cc.utah.edu/~dep04a60/wafer%20bonding.html.

7. **Lecarpentier, Gilbert and Vos., Joeri De.** Die 2 Die Bonding. *Die 2 Die Bonding.* SET S.A.S. (Smart Equipment Technology), 131 Impasse Barteudet, 74490 Saint Jeoire, France & IMEC, Kapeldreef 75, Leuven B-3001, Belgium : s.n., 2012.

8. *Computer Architecture for Die Stacking.* **Loh, G.** 2012, International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA), Vol. N/A, p. N/A.

9. *3D-Stacked Memory Architectures for Multi-core Processors.* **Loh, G.H.** 2008. Computer Architecture, 2008. ISCA '08. 35th International Symposium on. pp. 453-464. ISSN: 1063-6897 DOI: 10.1109/ISCA.2008.15.

10. *Process integration, device & structures.* **Raoux.** 2007, N/A, Vol. N/A, p. N/A.

11. *Phase-change random access memory: A scalable technology.* **Raoux, S., et al., et al.** 4.5, 2008, IBM Journal of Research and Development, Vol. 52, pp. 465-479. ISSN: 0018-8646 DOI: 10.1147/rd.524.0465.

12. **Quereshi.** *Phase Change Memory – from Devices to Systems.* [ed.] Morgan & Claypool Publishers. s.l. : Morgan & Claypool Publishers, 2011.

13. **Sherman, Jared, et al., et al.** A Multi-core Memory Organization for 3-D DRAM as Main Memory. [ed.] Hana KubÃ¡tovÃ¡, et al., et al. *Architecture of Computing Systems - ARCS 2013.* s.l. : Springer Berlin Heidelberg, 2013, Vol. 7767, pp. 62-73.

14. **Delattre, Frank.** Report: Intel Nehalem Architecture. *Report: Intel Nehalem Architecture.* 2008.

15. **Command, Linux.**

16. Pid in Microsoft Windows. *Pid in Microsoft Windows.*

17. *A performance comparison of contemporary DRAM architectures.* **Cuppu, V., et al., et al.** 1999. Computer Architecture, 1999. Proceedings of the 26th International Symposium on. pp. 222-233. ISSN: 1063-6897 DOI: 10.1109/ISCA.1999.765953.

18. *Energy efficient Phase Change Memory based main memory for future high performance systems.* **Bheda, R.A., et al., et al.** 2011. Green Computing Conference and Workshops (IGCC), 2011 International. pp. 1-8. DOI: 10.1109/IGCC.2011.6008569.

19. *A low-overhead coherence solution for multiprocessors with private cache memories.* **Papamarcos, Mark S. and Patel, Janak H.** 3, New York, NY, USA : ACM, #jan# 1984, SIGARCH Comput. Archit. News, Vol. 12, pp. 348-354. ISSN: 0163-5964 DOI: 10.1145/773453.808204.

20. **Various.** MESI Protocol. [Online] N/A. http://en.wikipedia.org/wiki/MESI_protocol.

21. PCMs. *PCMs.* 2013.

22. **Regina, Giuseppe, Pianelli, Stefano and Pisano, Giandomenico.** *Memory Architectures Solutions using 3D-DRAM as LLC and PCM as Main Memory.* University of Pisa. 2013. Master's thesis.

23. *CACTI: an enhanced cache access and cycle time model.* **Wilton, S. J E and Jouppi, N.P.** 5, 1996, Solid-State Circuits, IEEE Journal of, Vol. 31, pp. 677-688. ISSN: 0018-9200 DOI: 10.1109/4.509850.

24. *CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory*. **Chen, Ke, et al., et al.** 2012. Design, Automation Test in Europe Conference Exhibition (DATE), 2012. pp. 33-38. ISSN: 1530-1591 DOI: 10.1109/DATE.2012.6176428.

25. **River, Wind.** Simics. [Online] 2013.

APPENDICE A: File di Configurazione

In questa appendice si riportano i vari file di configurazione usati nei vari software per le simulazioni.

File di Configurazione per Cacti-3DD

Si riportano in questa sezione i vari file di configurazione per Cacti 3DD relativi alle configurazioni risultate migliori dopo la comparazione dei risultati.

2D-DRAM_8_GB_16Banks_4KBpP

File di configurazione relativo ad una 2D-DRAM (usata come *baseline*) della dimensione totale di 8 GB ripartiti su 16 differenti banchi e supponendo un'unità di pagina di 4 KB.

```
-size (Gb) 8
-block size (bytes) 256
-associativity 1
-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0
-UCA bank count 16
-technology (u) 0.032
-burst length 8
-internal prefetch width 8
-Data array cell type - "comm-dram"
-Data array peripheral type - "itrs-lstp"
-Tag array cell type - "itrs-hp"
-Tag array peripheral type - "itrs-hp"
-output/input bus width 64
-operating temperature (K) 350
-cache type "3D memory or 2D main memory"
-page size (bits) 4096
-burst depth 8
```

```
        -IO width 8

        -system frequency (MHz) 266

        -stacked die count 1

        -partitioning granularity 0

        -tag size (b) "default"

        -access mode (normal, sequential, fast) - "fast"

        -design objective (weight delay, dynamic power, leakage power, cycle time, area)
0:0:0:0:100

        -deviate (delay, dynamic power, leakage power, cycle time, area)
50:100000:100000:100000:1000000

        -NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area)
0:0:0:0:100

        -NUCAdeviate (delay, dynamic power, leakage power, cycle time, area)
10:10000:10000:10000:10000

        -Optimize ED or ED^2 (ED, ED^2, NONE): "NONE"

        -Cache model (NUCA, UCA)  - "UCA"

        -NUCA bank count 0

        -Wire signalling (fullswing, lowswing, default) - "Global_5"

        -Wire inside mat - "semi-global"

        -Wire outside mat - "global"

        -Interconnect projection - "conservative"

        -Core count 8

        -Cache level (L2/L3) - "L3"

        -Add ECC - "false"

        -Print level (DETAILED, CONCISE) - "DETAILED"

        -Print input parameters - "false"

        -Force cache config - "true"

        -Ndwl 128

        -Ndbl 32

        -Nspd 1

        -Ndcm 1

        -Ndsam1 1

        -Ndsam2 1
```

3D-DRAM_2GB_4Banks_32KBpP_4Dies

File di configurazione relative ad una memoria 3D sviluppata su 4 *die* e dotata di 2 GB di spazio d'archiviazione organizzata in 4 banchi.

Questo file di configurazione è inoltre la base, *mutatis mutandis*, per i seguenti altri file:

- 3D-DRAM_8GB_8Banks_32KBpP_4Dies
- 3D-DRAM_16GB_32Banks_32KBpP_4Dies
- 3D-DRAM_32GB_64Banks_32KBpP_4Dies

```
-size (Gb) 2
-block size (bytes) 256
-associativity 1
-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0
-UCA bank count 4
-technology (u) 0.032
-burst length 8
-internal prefetch width 8
-Data array cell type - "comm-dram"
-Data array peripheral type - "itrs-lstp"
-Tag array cell type - "itrs-hp"
-Tag array peripheral type - "itrs-hp"
-output/input bus width 64
-operating temperature (K) 350
-cache type "3D memory or 2D main memory"
-page size (bits) 32768
-burst depth 8
-IO width 8
-system frequency (MHz) 667
-stacked die count 4
-partitioning granularity 1
-tag size (b) "default"
-access mode (normal, sequential, fast) - "fast"
-design objective (weight delay, dynamic power, leakage power, cycle time, area) 0:0:0:0:100
-deviate (delay, dynamic power, leakage power, cycle time, area) 50:100000:100000:100000:1000000
```

```
	-NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area) 0:0:0:0:100
	-NUCAdeviate (delay, dynamic power, leakage power, cycle time, area) 10:10000:10000:10000:10000
	-Optimize ED or ED^2 (ED, ED^2, NONE): "NONE"
	-Cache model (NUCA, UCA)  - "UCA"
	-NUCA bank count 0
	-Wire signalling (fullswing, lowswing, default) - "Global_5"
	-Wire inside mat - "semi-global"
	-Wire outside mat - "global"
	-Interconnect projection - "conservative"

	-Core count 8
	-Cache level (L2/L3) - "L3"
	-Add ECC - "false"
	-Print level (DETAILED, CONCISE) - "DETAILED"
	-Print input parameters - "false"
	-Force cache config - "true"
	-Ndwl 128
	-Ndbl 32
	-Nspd 1
	-Ndcm 1
	-Ndsam1 1
	-Ndsam2 1
```

SRAM_Cache_IndexTrick_8GB_1Bank

Questo file di configurazione è la base per la simulazione di una SRAM capace di indirizzare il numero di pagine da 32 KB contenute in una DRAM da 8 GB. L'SRAM è inoltre suddivisa in un solo banco.

Il file di configurazione è inoltre la base, *mutatis mutandis*, per i seguenti altri file:

- SRAM_Cache_IndexTrick_16GB_2Banks
- SRAM_Cache_IndexTrick_32GB_2Banks

```
	-size (bytes) 262144
	-Array Power Gating - "true"
	-WL Power Gating - "true"
	-CL Power Gating - "true"
	-Bitline floating - "true"
	-Interconnect Power Gating - "true"
	-Power Gating Performance Loss 0.01
	-block size (bytes) 1
```

```
        -associativity 8
        -read-write port 1
        -exclusive read port 0
        -exclusive write port 0
        -single ended read ports 0
        -UCA bank count 1
        -technology (u) 0.022
        -page size (bits) 8192
        -burst length 8
        -internal prefetch width 8
        -Data array cell type - "itrs-hp"
        -Data array peripheral type - "itrs-hp"
        -Tag array cell type - "itrs-hp"
        -Tag array peripheral type - "itrs-hp"
        -output/input bus width 22
        -operating temperature (K) 360
        -cache type "cache"
        -tag size (b) 7
        -access mode (normal, sequential, fast) - "normal"
        -design objective (weight delay, dynamic power, leakage power, cycle time, area)
0:0:0:100:0
        -deviate (delay, dynamic power, leakage power, cycle time, area)
20:100000:100000:100000:100000
        -NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area)
100:100:0:0:100
        -NUCAdeviate (delay, dynamic power, leakage power, cycle time, area)
10:10000:10000:10000:10000
        -Optimize ED or ED^2 (ED, ED^2, NONE): "ED^2"
        -Cache model (NUCA, UCA)  - "UCA"
        -NUCA bank count 0
        -Wire signalling (fullswing, lowswing, default) - "Global_30"
        -Wire inside mat - "semi-global"
        -Wire outside mat - "semi-global"
        -Interconnect projection - "conservative"
        -Core count 8
        -Cache level (L2/L3) - "L3"
        -Add ECC - "true"
        -Print level (DETAILED, CONCISE) - "DETAILED"
        -Print input parameters - "false"
        -Force cache config - "false"
        -Ndwl 1
        -Ndbl 1
        -Nspd 0
        -Ndcm 1
```

```
-Ndsam1 0
-Ndsam2 0
```

File di Configurazione per Cacti-TLB

Cacti-TLB è una versione non definitiva di Cacti (versione 7.0) appositamente richiesta ed ottenuta dall'autore per la simulazione di cache *fully-associative* e, in particolare, per la simulazione di un TLB *fully-associative* durante la comparazione dei vari TLB da affiancare alla CMM. Il seguente è il file di configurazione del miglior TLB esaminato (TLB_WA8_1024Entries):

```
-size (bytes) 26624
-Array Power Gating - "true"
-WL Power Gating - "true"
-CL Power Gating - "true"
-Bitline floating - "true"
-Interconnect Power Gating - "true"
-Power Gating Performance Loss 0.01
-block size (bytes) 26
-associativity 8
-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0
-UCA bank count 1
-technology (u) 0.022
-page size (bits) 8192
-burst length 8
-internal prefetch width 8
-Data array cell type - "itrs-hp"
-Data array peripheral type - "itrs-hp"
-Tag array cell type - "itrs-hp"
-Tag array peripheral type - "itrs-hp"
-output/input bus width 152
-operating temperature (K) 370
-cache type "cache"
-tag size (b) 56
-access mode (normal, sequential, fast) - "normal"
-design objective (weight delay, dynamic power, leakage power, cycle time, area) 0:0:0:100:0
-deviate (delay, dynamic power, leakage power, cycle time, area) 20:100000:100000:100000:100000
```

```
        -NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area)
100:100:0:0:100

        -NUCAdeviate (delay, dynamic power, leakage power, cycle time, area)
10:10000:10000:10000:10000

        -Optimize ED or ED^2 (ED, ED^2, NONE): "ED^2"

        -Cache model (NUCA, UCA)  - "UCA"

        -NUCA bank count 0

        -Wire signalling (fullswing, lowswing, default) - "Global_30"

        -Wire inside mat - "semi-global"

        -Wire outside mat - "semi-global"

        -Interconnect projection - "conservative"

        -Core count 8

        -Cache level (L2/L3) - "L3"

        -Add ECC - "true"

        -Print level (DETAILED, CONCISE) - "DETAILED"

        -Print input parameters - "false"

        -Force cache config - "false"

        -Ndwl 1

        -Ndbl 1

        -Nspd 0

        -Ndcm 1

        -Ndsam1 0

        -Ndsam2 0
```

APPENDICE B: File Sorgenti

In questa appendice sono riportati stralci dei codici sorgenti utilizzati per le simulazioni.

File: cmm-tlb.h

```
struct CmmTlbEntry
{
	uint8_t IsValid;
	uint64_t VirtualPageNumber;
	uint64_t ContextId;
	int LineNumber;
	uint8_t Counter;
};

typedef struct CmmTlbEntry cmm_tlb_entry_t;

struct Tlb
{
	uint32_t Entries;
	int Associativity;
	uint32_t Count;
	cmm_tlb_entry_t* Set;
};

typedef struct Tlb cmm_tlb_t;

int IsEmpty(cmm_tlb_t *tlb);
int GetNumberOfEntries(cmm_tlb_t *tlb);

int GetSetIndex(cmm_tlb_t *tlb, uint64 vpn);

void enqueue(cmm_tlb_t *tlb, cmm_tlb_entry_t *newEntry);
int find_entry(cmm_tlb_t *tlb, uint64 virtual_page_number, uint64 context_id);
cmm_tlb_entry_t *retrieveEntry(cmm_tlb_t *tlb, uint64 location, uint64 vpn, uint64
cid);
void update_newEntry_line_number(cmm_tlb_t *tlb, uint64 virtual_page_number,
uint64 context_id, int line_number);
void requeue_entry(cmm_tlb_t *tlb, uint64 location);
void remove_entry(cmm_tlb_t *tlb, uint64 vpn, uint64 cid);
void flush_tlb(cmm_tlb_t *tlb);
cmm_tlb_entry_t *create_new_tlb_entry(uint64 virtual_page_number, uint64
context_id, int line_number);

void initialize_tlb(cmm_tlb_t *tlb, uint64 max_size);
void finalize_tlb_init(cmm_tlb_t *tlb, int associativity);
```

File: Input8G16B4D.h

```
#ifndef _INPUT_8G_16B_4D_
#define _INPUT_8G_16B_4D_
/// Clock frequency of the processor in GHz
#define CLOCK_FREQUENCY 3
/// The memory size (in GB)
#define MEMORY_SIZE 8LL
/// The number of stacked dies (control logic die is not counted)
#define N_DIES 4
/// The total number of banks, belongin to different ranks, which are placed on_
the same die
#define N_BANKS_PER_DIE 16
/// The total number of rowbuffer per each set
#define N_ROWBUFFER_PER_SET 5
/// The number of coloumns in each bank. Each of them contains WORD_SIZE bits of_
data
#define N_COLOUMNS 256
/// Row to Column command delay (in ns)
#define t_RCD 14.2819
/// Time between column command and data out (in ns)
#define t_CAS 8.93226
/// Time between column commands (in ns)
/// NB: THIS IS NOT USED
#define t_CCD 0
/// Time to precharge DRAM array (in ns)
#define t_RP 3.23195
/// Time between RAS and data restoration in DRAM array (minimum time a row must_
be open) (in ns)
#define t_RAS 16.4724
/// Time the SRAM takes (in ns) to translate the virtual address if there has been
a TLB miss
#define t_SRAM 0.897256
/// Time the TLB takes (in ns) to give hit or miss
#define t_TLB 0.226757
/// The activation energy in this configuration
#define ACTIVATION_ENERGY 0.686793
/// The energy consumed during a read operation
#define READ_ENERGY 0.805576
/// The enrgy consumed during a write operation
#define WRITE_ENERGY 0.885578
/// The enrgy for a precharge
#define PRECHARGE_ENERGY 0.591437
/// The Energy consumed by SRAM
#define SRAM_ENERGY 0.01073225
// The energy consumed for accessing and reading TLB
#define TLB_READ_ENERGY 0.00821664
// The energy consumed for accessing and writing TLB
#define TLB_WRITE_ENERGY 0.0129158

#endif
```

File: Memory.h

```
#ifndef _MEMORY_H_
#define _MEMORY_H_

#include <stdint.h>
#include <math.h>
#include "Input.h"

#ifndef _Bool // Should be compatible with C99
#define _Bool int
#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif
#else
#include <stdbool.h>
#endif

#define MEMORY_SIZE_IN_BYTES (MEMORY_SIZE * 1073741824)
#define MEMORY_SIZE_IN_BITS (MEMORY_SIZE_IN_BYTES * 8)
#define N_RANKS 4
#define N_BANKS_PER_RANK N_BANKS_PER_RANK_PER_DIE * N_DIES
#define N_BANKS_PER_RANK_PER_DIE N_BANKS_PER_DIE / N_RANKS
#define N_BANKS N_BANKS_PER_DIE * N_DIES
#define N_ROWBUFFERS_SET N_BANKS_PER_DIE
#define N_ROWBUFFERS N_ROWBUFFERS_SET * N_ROWBUFFERS_PER_SET
#define WORD_SIZE 64 / N_DIES
#define N_ROWS (MEMORY_SIZE_IN_BITS / N_BANKS) / (N_COLOUMNS * WORD_SIZE)
#define ROWBUFFER_SIZE N_COLOUMNS * N_DIES
#define ROWBUFFER_SIZE_IN_BITS ROWBUFFER_SIZE * WORD_SIZE
#define t_RC t_RAS + t_RP
#define HIT_TIME t_CAS
#define SEMI_HIT_TIME HIT_TIME + t_RAS
#define SEMI_MISS_TIME t_RP + t_RCD + t_CAS
#define MISS_TIME t_RC + t_RCD + t_CAS
#define SRAM_ACCESS_DELAY t_SRAM * CLOCK_FREQUENCY

struct RowBuffer;
typedef struct RowBuffer RowBuffer;

enum MemoryOperation
{
	READ = 0,
	WRITE = 1,
	SRAM_RWDATA_HIT = 2,
	SRAM_RWDATA_MISS = 4,
};

typedef enum MemoryOperation MemoryOperation;

uint16_t GetClockCycleFromDelay(double time);
double GetDelayFromClockCycle(uint16_t clockCycles);
uint16_t GetDRAMDelay(MemoryOperation operation, uint64_t address, uint64_t
clockCycles);
```

```
uint16_t GetReadDelay(uint64_t address, uint64_t clockCycles);
uint16_t GetWriteDelay(uint64_t address, uint64_t clockCycles);
uint16_t GetSRAMDelay();
double GetDRAMConsumedEnergy(MemoryOperation operation, uint16_t delay, double*
buffer);
double GetSRAMConsumedEnergy(MemoryOperation operation, double* buffer);

#endif
```

IN SUPREMÆ DIGNITATIS
1343