PROGETTAZIONE E VALUTAZIONE DI UN SOTTOSISTEMA DI MEMORIA PRINCIPALE AD ALTE PRESTAZIONI BASATO SU 3D-DRAM E PCM : MODELLAZIONE, STRESS-TESTING ED ANALISI DELLE PERFORMANCE E DEL CONSUMO ENERGETICO

STEFANO PIANELLI

Primo relatore : Prof. Cosimo Antonio Prete
Secondo relatore : Prof. Pierfrancesco Foglia
Supervisore : Prof. Krishna Kavi

06 Giugno 2013

Facolta' di Ingegneria
Dipartimento di Ingegneria Informatica
Universita' di Pisa

Se la pioggia vi sorprende a metà strada,
e camminate più in fretta per trovare un riparo,
nel passare sotto alle grondaie o nei punti scoperti,
vi bagnerete ugualmente.
Se invece ammettete sin dall'inizio la possibilità di bagnarvi,
non vi darete pena, pur bagnandovi lo stesso.

— Tsunetomo Yamamoto

SOMMARIO

Negli ultimi anni la progettazione e i processi produttivi dei componenti di memoria hanno subito un forte rallentamento, molto accentuato per quanto riguarda quei componenti utilizzati nei sottosistemi di memoria principale, come le memorie DRAM. Un grande divario prestazionale si è venuto a creare tra le unità di elaborazione, che hanno subito un vertiginoso *speed-up* negli ultimi 10 anni, e i supporti di memoria che ormai a fatica riescono a fronteggiare le richieste del mercato, fortemente incentrato sul *cloud-computing* e su grandi sistemi di elaborazione distribuiti che hanno bisogno di grandi quantità di memoria.

L'intento di questo lavoro è quello di progettare e sperimentare alcune architetture di memoria alternative a quella classica per cercare di trovare una valida soluzione al problema sopra evidenziato, concentrando l'attenzione sugli aspetti prestazionali e sul consumo energetico, pur avendo come obiettivo principale quello di mantenere la *scalabilità* dell'architettura punto focale del nostro studio.

A tal fine sono state esplorate nuove promettenti tecnologie, quali i dispositivi 3D-DRAM e PCM (*Phase Change Memory*), inserite in un costesto architetturale del tutto rinnovato.

Al fine di rendere lo studio delle architetture il più concreto possibile sono stati realizzati modelli dettagliati delle memorie sopra citate, andando ad analizzare i *trade-off* tra le velocità delle operazioni di memoria e il consumo energetico a queste legate.

La valutazione finale delle architetture è stata effettuata inserendo queste in un contesto reale simulato e sono stati effettuati *stress-test* utilizzando note suite di benchmark quali *SPEC CPU2006* e *OLTP*, entrambe rivolte allo studio di sistemi ad alte prestazioni e sistemi *HPC* (*High Performance Computers*) che, come abbiamo detto, richiedono una notevole quantità di memoria, prestazioni elevate e consumi contenuti.

Lo studio ha portato ad evindenziare un incremento di prestazioni fino al 40% e consumi pressochè paragonabili se confrontati con le odierne architetture di memoria principale.

Ho sempre avuto pochissime idee,
ma in compenso fisse.

— Fabrizio De Andrè

RINGRAZIAMENTI

Ringrazio coloro che mi hanno sempre dato tutto,

un amore incommensurabile, una vita splendida;

coloro che mi hanno infuso una fervida curiosità

accompagnata da una accecante fantasia;

coloro che con ferma severità mi hanno sempre spinto avanti,

pur tenendomi la mano in ogni momento.

I miei genitori.

INDICE

I INTRODUZIONE 1

1 3D-DIE STACKING 3
1.1 Struttura tridimensionale 3
1.2 3D-Stacked DRAM 4
1.2.1 Organizzazione e funzionamento di una classica memoria 2D DRAM 4
1.2.2 Organizzazione e funzionamento di una 3D-Stacked DRAM 5

2 PCM - PHASE CHANGE MEMORY 9
2.1 Cella di memoria PCM 9

II ARCHITETTURE PROPOSTE 13

3 CACHE MAIN MEMORY 15
3.1 Organizzazione della memoria CMM 15
3.2 Cache di primo e di secondo livello 16
3.3 Strutture hardware di supporto all'architettura CMM 17
3.3.1 TLB 17
3.3.2 Bitmap 18
3.3.3 SRAM 18
3.3.4 DRAM 22
3.3.5 Row buffers 22
3.3.6 Rimpiazzamento dei dati nella SRAM 23
3.3.7 Victim Cache 23
3.3.8 FRTD come Back Storage 24

4 PCM COME MAIN MEMORY E 3D-DRAM COME LLC 25
4.1 PCM come Main Memory 25
4.1.1 Organizzazione di una memoria principale realizzata tramite PCM 26
4.2 PCM come Storage System 26
4.3 Presente e futuro delle PCM 27
4.4 3D-DRAM come Last Level Cache 28
4.4.1 Coerenza della 3D-DRAM come LLC 28
4.5 Caratteristiche della 3D-LLC 30
4.5.1 SRAM 30
4.5.2 DRAM 31
4.5.3 Row-Buffers 32
4.6 Caratteristiche della PCM as Main Memory 32
4.6.1 Dimensionamento delle sottopagine 33
4.6.2 Rowbuffers 34
4.7 Panoramica riassuntiva dell'architettura 3D-DRAM as LLC e PCM as Main Memory 34

5 PCM COME MM E 3D-DRAM COME PARTE DELLA MM 37

5.1 Politiche di smistamento Hardware-based 38
5.1.1 Politica di smistamento a gruppi 38
5.1.2 Politica di smistamento a gruppi alternativa 40
5.1.3 Politica di smistamento forte 41
5.1.4 Politica di smistamento contiguo 43
5.1.5 Politica di smistamento basata sugli accessi alle pagine 43
5.2 Politiche kernel-based 44

III STRUMENTAZIONI DI SIMULAZIONE 45

6 SIMULATORE HP CACTI 47
6.1 Parametri di configurazione di CACTI 47
6.1.1 Parametri di configurazione per entrambi tipi di memoria 48
6.2 Parametri di configurazione per memorie cache 50
6.2.1 Parametri di configurazione per memorie principali 51
6.3 File output di CACTI 51
6.3.1 Componenti di output per memorie cache 52
6.3.2 Componenti di output per memorie principali 53

7 CACTI-3DD 57

8 WIND RIVER SIMICS 61

9 BENCHMARK SUITES 65
9.1 SPEC CPU2006 65
9.2 OLTP 69

IV SIMULAZIONI E ANALISI DEI RISULTATI 71

10 EXPERIMENTAL SETUP 73

11 SIMULAZIONI ED ANALISI DEI RISULTATI OTTENUTE CON CACTI 75
11.1 2D DRAM BASELINE 75
11.2 TLB - Tranlsation Lookaside Buffer 78
11.3 SRAM 80
11.4 3D DRAM 83

12 PARAMETRI DI PROGETTAZIONE OTTENUTI DA ALTRI STUDI 87

13 PCM COME MM E 3D-DRAM COME PARTE DELLA MM 89
13.1 Modifica dell'ambiente di simulazione 91
13.1.1 Note realizzate del bus di memoria 93
13.2 Analisi dei risultati 93
13.2.1 Suite SPEC CPU2006 : IPC 95
13.2.2 Suite SPEC CPU2006 : consumo di potenza 96
13.2.3 Suite OLTP : IPC 97
13.2.4 Suite OLTP : consumo di potenza 98
13.3 Sviluppi futuri 99

BIBLIOGRAFIA 101

ELENCO DELLE FIGURE

Figura 1 Struttura 3D raffigurante due *die* sovrapposti, da [7] 3
Figura 2 Organizzazione di una generica DRAM, da [7] 4
Figura 3 Organizzare di un'architettura *Wide-3D*, da [7] 6
Figura 4 Organizzazione di un'architettura *True-3D*, da [7] 7
Figura 5 Visione al microscopio delle due fasi del materiale calcolgenico, da [11] 10
Figura 6 Cella elementare di memoria PCM e tempi delle operazioni, da [11] 11
Figura 7 Schema logico dell'architettura CMM 16
Figura 8 Struttura interna della SRAM 21
Figura 9 Algoritmo completo di accesso alla CMM 23
Figura 10 Latenze di accesso tipiche per varie tecnologie (in cicli di clock), da [12] 25
Figura 11 Architettura semplificata con LLC 29
Figura 12 Architettura di un processore i7 di prima generazione (Nehalem) 30
Figura 13 Struttra del sistema 3D-DRAM as LLC 32
Figura 14 Visione di insieme del sistema 3D-DRAM-LLC + PCM-MM 35
Figura 15 Architettura ibrida di memoria 3D-DRAM e PCM 38
Figura 16 Architettura ibrida di memoria 3D-DRAM e PCM 38
Figura 17 Indirizzo fisico in uscita dal sistema CMM 39
Figura 18 Algoritmo di smistamento dei dati nella memoria ibrida. Da notare come, per semplicità, è stata omessa in figura la sottrazione dal resto della divisione intera del numero di pagine occupate dalla 3D DRAM nel singolo gruppo 40
Figura 19 Politica Random Mixed alternative 41
Figura 20 Politica di smistamento hard 42
Figura 21 Politica di smistamento Random Continuous 43
Figura 22 Diagramma a blocchi del framework di CACTI-3DD 57
Figura 23 Modello architetturale per *WIDE-3D* e *TRUE-3D Stacked DRAM*, da [2] 58
Figura 24 Evoluzione dal *planar-bank* allo *Stacked-bank*, da [2] 58
Figura 25 Framework di simulazione di Wind River SIMICS 61
Figura 26 Delay (ns) per le componenti di accesso alla DRAM 76

Figura 27 Energia (nJ) consumata per un singolo accesso in memoria 76
Figura 28 Area (mm^2) delle varie memorie 77
Figura 29 Confronto fra i tempi di accesso al TLB 79
Figura 30 Confronto tra le le energia consumate dagli accessi in TLB 79
Figura 31 Cofronto tra le aree occupate dai TLB 79
Figura 32 Struttura finale di una entrata in SRAM 81
Figura 33 Latenze per le varie SRAM 82
Figura 34 Energia consumata da un singolo accesso nelle varie SRAM 82
Figura 35 Latenze di accesso medie per ogni tipo di memoria considerate 84
Figura 36 Energia consumata durante un accesso della durata media in DRAM 84
Figura 37 Algoritmo di smistamento a gruppi 90
Figura 38 Flow-chart dell'architettura realizzata in Simics 91
Figura 39 IPC per la suite SPEC CPU2006 95
Figura 40 Potenza consumata per la suite SPEC CPU2006 96
Figura 41 IPC per la suite OLTP 98
Figura 42 Potenza consumata per la suite OLTP 99
Figura 43 Percentuale di smistamento tra 3D DRAM e PCM degli accessi in memoria 100

ELENCO DELLE TABELLE

Tabella 1 Singola linea di cache L1/L2 16
Tabella 2 Singola entry del TLB 17
Tabella 3 Singola entry del TLB comprendente la *bitmap* 18
Tabella 4 Struttura di un indirizzo virtuale nell'architettura CMM 19
Tabella 5 Struttura *TAG/Location* 20
Tabella 6 Struttura di un indirizzo fisico 21
Tabella 7 Struttura definitiva di un indirizzo fisico 22
Tabella 8 Struttura di una linea di cache nella SRAM della 3DLLC 31
Tabella 9 OLTP Benchmark 73
Tabella 10 Tipi di memoria analizzati per la determinazione della Baseline 75
Tabella 11 Tipi di TLB simulati con Cacti 78
Tabella 12 Lista delle varie configurazioni di SRAM testate 81
Tabella 13 Varianti della 3D DRAM simulate in CACTI 83

Tabella 14 Valori di progetto ottenuti per la PCM 88
Tabella 15 Benchmark mixes e relativi workloads per la suite SPEC CPU2006 94
Tabella 16 Benchmark mixes e relativi workloads per la suite OLTP 94
Tabella 17 Configurazioni hardware testate con la suite SPEC CPU2006 95
Tabella 18 Configurazioni hardware testate con la suite OLTP 97

ELENCO DEGLI ALGORITMI

Figura 5.1 Algoritmo per il calcolo dell'indirizzo fisico ibrido di memoria nella politica di smistamento forte 42
Figura 13.1 Algoritmo estratto dal modulo **Splitter** che effettua lo smistamento e la traduzione dell'indirizzo fisico nell'indirizzo fisico ibrido di memoria 92

ELENCO DEI SIMBOLI

3D	Three Dimensional
3D-LLC	3D-DRAM as Last Level Cache
ASID	Application Specific IDentifier
CAM	Content Addressable Memory
CMM	Cache Main Memory
CMOS	Complementary MOS
D2D	Die to Die
FIFO	First In First Out
FRTD	Flash Replacement Technology Drive
FTL	Flash Translation Layer
IC	Integrated Circuit
L1	Level 1 [Cache]

L2 Level 2 [Cache]

L3 Level 3 [Cache]

LLC Last Level Cache

LRU Least Recently Used

MC Memory Controller

MESI Modified Exclusive Shared Invalid [Protocol]

MOS Metal-Oxide-Semiconductor

MRQ Memory Request Queue

N-MOS N-Type MOS

NUMA Non-Uniform Memory Access

P-MOS P-Type MOS

PA Physical Address

PCM Phase Change Memory

PCM-MM Phase Change Memory as Main Memory

PMM [3D-DRAM as] Part of Main Memory

SATA Serial ATA (Serial AT Attachment)

SCM Storage Class Memory

SCSI Small Computer System Interface

SRAM Static Random Access Memory

SSD Solid State Drive

TLB Translation Lookaside Buffer

TSV Through Silicon Via

UMA Uniform Memory Access

VA Virtual Address

Parte I

INTRODUZIONE

La sempre maggiore velocità dei processori, nonostante consenta un maggior numero di istruzioni per unità di tempo, accentua sempre più un problema noto in letteratura come *Memory Wall*, in [15], ovvero il grande divario che intercorre tra la velocità con cui i dati vengono richiesti dalla unità di elaborazione e la velocità con cui questi possano essere serviti dalle unità di memorizzazione. Al fine di ridurre questo grande divario 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. Lo scopo di questo documento è dimostrare come un approccio totalmente diverso basato su moderni supporti di memorizzazione come 3D-DRAM e PCM e su diversi meccanismi di indirizzamento possa tentare di diminuire il divario sopra citato.

1

3D-DIE STACKING

Con il termine *3D Die Stacking* si intende un nuovo tipo di tecnologia che permette l'integrazione verticale di due o più *die* i quali comunicano tra di loro attraverso un'interfaccia ad alta densità e velocità. Questo tipo di tecnologia offre inoltre la possibilità di poter realizzare diversi tipi di *die*, con tecnologie diverse tra loro, da poter integrare su di un unico *stacked die*, permettendo così un forte rilassamento dei vincoli di progettazione e un potenziale incremento prestazionale sotto ogni punto di vista : velocità operazionale e consumo energetico. Poter infatti scegliere la migliore tecnologia per ogni *die* che si vuole integrare nella struttura tridimensionale è un grande punto a favore per la risoluzione di ovvi problemi di ottimizzazione.

Tuttavia adottare questo tipo di tecnologia pone i progettisti di fronte a problematiche non trascurabili, quali lo studio attento dell'interfaccia di comunicazione tra i diversi *die* e il più immediato problema legato allo smaltimento del calore all'interno della struttura tridimensionale.

1.1 STRUTTURA TRIDIMENSIONALE

Nell'ambito di un circuito integrato le piste metalliche sono una fonte primaria di overhead in termini di latenza, area ed energia consumata. Diversi studi, tra i quali quelli in [1], dimostrano che nelle varie connessioni all'interno di un microprocessore le piste metalliche partecipano fino ad un massimo 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 *die*, dette *Through Silicon Via (TSV)*.

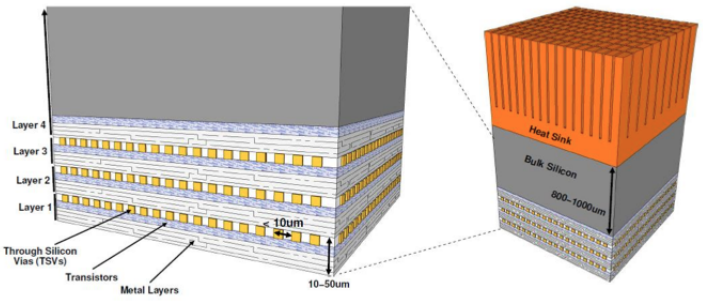

Figura 1: Struttura 3D raffigurante due *die* sovrapposti, da [7]

Vi sono diverse metodologie per la sovrapposizione di più *die*, tra

i quali ricordiamo il *wafer-to-wafer bonding*[1], *die-to-die bonding* e *die-to-wafer bonding* ampiamente trattati in [6], e dei quali non entreremo nel dettaglio implementativo.

Nel seguito di supporrà una metodologia *die-to-die* con sovrapposizione *face-to-face*.

1.2 3D-STACKED DRAM

La tecnologia *3D Stacked DRAM* è stata presentata alla comunità scientifica 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 interconnesioni *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 studi in [6] 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 [7] e verrà esposta nei paragrafi successivi.

1.2.1 *Organizzazione e funzionamento di una classica memoria 2D DRAM*

Una memoria DRAM può essere vista come una matrice di *bitcells* ciascuna delle quali implementata tramite un unico trasistor, affiancata da logica accessoria necessaria alla gestione delle operazioni di memoria.

La figura successiva illustra una generica gerarchia di memoria.

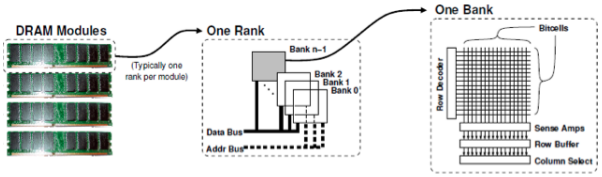

Figura 2: Organizzazione di una generica DRAM, da [7]

Cominciando ad analizzare a più alto livello una memoria DRAM, questa risulta divisa in ranghi o *ranks*, tipicamente uno o due per modulo.

1 http://www.cc.utah.edu/~dep04a60/wafer%20bonding.html

All'interno di ciascun *rank* la memoria è suddivisa in banchi (*banks*). Ogni *bank* risulta essere in ultima analisi una matrice di *bitcell*.

Durante un'operazione di lettura in DRAM parti differenti dell'indirizzo fisico di memoria vengono utilizzate per selezionare *rank*, *bank* per ogni modulo e infine la riga (*row*) all'interno della matrice di *bitcell*. Un insieme di *sense amplifiers* legge i valori memorizzati in ogni unità della riga selezionata registrandone il contenuto in un *row buffer*. I rimanenti bit dell'indirizzo fisico di memoria selezionano infine porzione di dato all'interno del *row buffer*; questa ultima operazione sancisce la fine di una richiesta di memoria e il dato è reso disponibile al richiedente.

È 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 transistor di passo che scaricano la carica elettrica contenuto nelle *bitcell*. I *sense amplifier* registrano le variazioni di corrente sulla bitline e forniscono in uscita il dato memorizzato.

Inoltre si noti come periodicamente il contenuto della DRAM vada 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 *rank* o *bank* aumentando così il parallelismo negli accessi in memoria, tenendo sempre in considerazione problematiche quali la limitazione nella densità di transistor in ogni modulo di memoria per problemi energetici e di dissipazione del calore e per problematiche legate alla superficie disponibile su ogni modulo che risulta limitata.

1.2.2 *Organizzazione e funzionamento di una 3D-Stacked DRAM*

Nel seguito verranno presentati due differenti approcci per la realizzazione di un'architettura di memoria 3D-DRAM: *Wide-3D* e *True-3D*.

Approccio Wide-3D

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

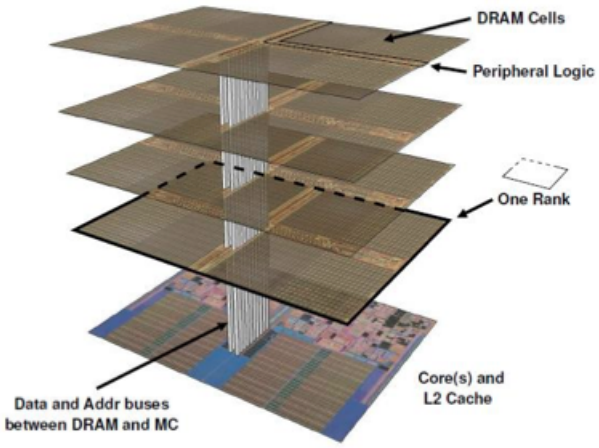

Figura 3: Organizzare di un'architettura *Wide-3D,* da [7]

Questo approccio prevede N *stacked layers* ognuno corrispondente ad un *die*; il *layer* di indice 0 risulta essere l'unità operativa centrale, la *CPU*, mentre gli $N-1$ layer superiori contengono sia logica di controllo ed amplificazione come *row buffers*, *sense amplifiers* (vista la disponibilità di area all'interno dei chip *stacked* è possibile posizionare nel *layer* subito superiore la *CPU* unità operative quali *cache* di livello superiore come *Last Level Caches* oppure *TLB*) sia le vere e proprie matrici di *bitcell*.

Il singolo bus centrale, noto come *stripe bus*, è condiviso tra i vari layer e risulta essere un collo di bottiglia per transazioni parallele.

È semplice constatare che questo approccio risulta essere una riproposizione, in formato 3D, dei vari moduli/ranghi/banchi presenti nell'approccio planare e del quale si è discusso in precedenza, in cui al posto di avere una memoria *off chip* si ha a che fare con una memoria *stacked* sopra il processore e un bus più corto e più veloce costituito dalle *TSV*.

Tra gli approcci sopra menzionati, il *Wide-3D* è l'unico ad essere stato effettivamente prototipato al giorno d'oggi.

Approccio TRUE-3D

La figura 4 mostra l'organizzazione di una *True-3D DRAM*.

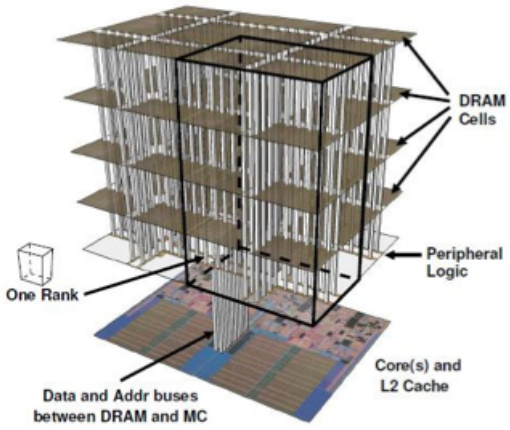

Figura 4: Organizzazione di un'architettura *True-3D*, da [7]

Anche in questo apporccio il *layer* di indice 0 è costituito dall'unità operativa centrale. Il *layer* di indice 1 è stato progettato per contenere tutta la logica di controllo aggiuntiva necessaria per la gestione della memoria, come *rowbuffer*, *memory controller*, in aggiunta alle altre strutture di supporto al sistema già menzionate nel capitolo precedente. La scelta di 'spostare' la logica di controllo dai *die* di memoria ad un *die* dedicato è stata fatta per poter rilassare i vincoli di costruzione dei *die* stessi in termini di area disponibile, consumo di potenza e dissipazione di calore, potendo così utilizzare per questi processi produttivi indirizzati verso una più alta efficienza. E' noto infatti che per l'ottimizzazione rispettivamente di logica di memoria e logica di controllo si utilizzano processi produttivi completamente diversi.

Gli $N-2$ layer superiori contengono le matrici di *bitcells* e sono realizzati con processi tecnologici ottimizzati per la densità. L'approccio *die stacked* inoltre è sfruttato intensamente anche sul generico layer : i singoli *rank* di memoria sono a loro volta realizzati *stacked* ovvero i *banks* che vanno a formare un *rank* non appartengono più solamente ad un unico *die*, ma sono a loro volta distribuiti su *dies* adiacenti. A loro volta i singoli *banks* contenuti in ogni die sono suddivisi in *sub-banks* posizionati in pila tra di loro su ogni singolo *die*. Una descrizione più approfondita di questo ultimo approccio verrà presentata nel capitolo contenente i dettagli di modellazione utilizzati per la simulazione dell'architettura.

Un altro grande punto di forza di questo approccio consiste nella suddivisione dell'unico bus centrale in vari *'sub-bus'* ciascuno dei quali relativo ad un singolo *rank* tridimensionale: in questo modo ogni singolo *rank* di memoria potrà essere acceduto parallelamente grazie all'utilizzo della tecnologia *TSV* per le vie di comunicazione *inter-die* e per la comunicazione tra *die* di memoria e unità di elaborazione centrale.

2 PCM - PHASE CHANGE MEMORY

Lo *scaling* del processo produttivo nella fabbricazione di memorie, obbiettivo di primaria importanza nell'industria microelettronica, è attualmente in una fase di stallo in quanto in quanto i processi produttivi di ultima generazione non riescono ormai da qualche anno a tenere testa, soprattutto nelle tecnologie più diffuse come ad esempio nelle memorie DRAM, al vertiginoso sviluppo che hanno che ha caratterizzato le unità di elaborazione.

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 che rimane costante nel tempo e che viene rilevato durante le operazioni di lettura. Il dispositivo PCM dunque non necessita di immagazzinare cariche elettriche. In quest'ottica, la corrente di programmazione, necessaria al cambiamento di fase nel materiale utilizzato per la costruzione delle celle di memoria PCM, scala linearmente.

Questo *scaling* lineare è stato dimostrato in un prototipo di dispositivo PCM facente uso di una tecnologia a $20nm$, in [13].

La PCM introduce comunque anche diversi inconvenienti, il più evidente dei quali riguarda le latenze d'accesso che, nonostante varino da poche decine a centinaia di nanosecondi, risultano essere almeno 2 ordini di grandezza maggiori di quelle relative ad una classica memoria DRAM. Allo stato attuale dell'arte le operazioni di scrittura richiedono una intensa e graduale iniezione di corrente, il che implica un forte consumo di energia.

Ulteriore punto debole di questa tecnologia risiede nel numero limitato di scritture che è possibile eseguire sul dispositivo. Ogni operazione di scrittura, in quanto ad alta corrente, induce espansioni e contrazioni termiche all'interno del materiale costitutivo delle celle di memorie PCM, degradandone l'integrità e limitandone la durata a centinaia di milioni di scritture.

2.1 CELLA DI MEMORIA 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 *calcogenico*, visibile in figura 5.

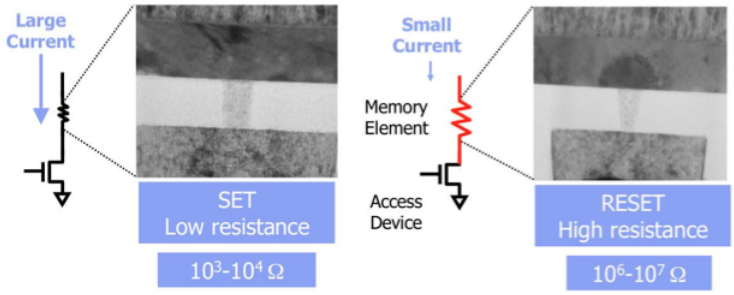

Figura 5: Visione al microscopio delle due fasi del materiale calcolgenico, da [11]

Il materiale calcogenico 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 portarlo ad una temperatura voluta ed ottenere conseguentemente un grado di impedenza noto che provvederà a memorizzare il dato: durante una lettura infatti si misura la tensione ai capi della cella di memoria attraversata da una corrente. Nei due 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 calcogenico può essere variata su diversi valori piuttosto distinti tra loro è possibile memorizzare fino a 4 bit in una singola cella di memoria, aumentando dunque la densità di memorizzazione di un fattore 8.

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.

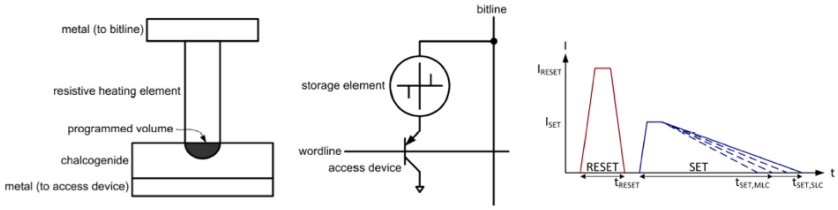

Figura 6: Cella elementare di memoria PCM e tempi delle operazioni, da [11]

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 calcogenico poco al di là della sua temperatura di fusione ($\sim 660°C$).

La densità di corrente richiesta per quest'operazione è nell'ordine dei $\frac{5 \div 20 MA}{cm^2}$. 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, operazione di *SET*, è anch'essa basata su effetto Joule. Per settare una cella 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 6.

Parte II

ARCHITETTURE PROPOSTE

Partendo dunque dalle tecnologie nella loro forma base, ossia 3D-DRAM nella versione *True-3D* e PCM è possibile espandere tali tecnologie creando architetture del tutto nuove ed innovative. In particolare si proporranno tre architetture che fanno uso, in maniera più o meno accentuata, sia delle memorie 3D DRAM sia delle Phase Change Memories: tali memorie sono la *Cache Main Memory*, in [5], *3D-DRAM come Last Level Cache* (a cui ci riferirà nel seguito con l'abbreviazione *3D-LLC*) e *3D-DRAM* come parte della memoria principale.

3 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 in [5]. Tale precedente studio infatti privilegiava aspetti relativi alla velocità di esecuzione e risposta di un sistema dotato di CMM rispetto ad un sistema con una classica e ormai consolidata architettura di memoria 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 e trattate nei capitoli seguenti.

3.1 ORGANIZZAZIONE DELLA MEMORIA CMM

La grande novità introdotta dall'architettura CMM è l'indirizzamento *cache-like* della memoria principale (costituita in questo studio da una memoria di tipo 3D DRAM), da cui deriva il nome stesso dell'architettura. Questo tipo di memoria corrisponde in tutto e per tutto ad una memoria principale ma si vedrà come grazie all'ausilio di un meccanismo di traduzione degli indirizzi innovativo e grazie all'utilizzo di componenti hardware di supporto all'indirizzamento *cache-like* si otterrano risultati molto convincenti se paragonati ad un approccio di memoria classico.

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 in figura 7, che come è facile notare appare del tutto uguale ad una classica architettura di memoria dei sistemi odierni :

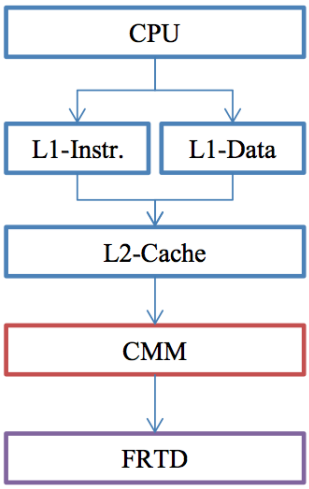

Figura 7: Schema logico dell'architettura CMM

I moduli CPU, Cache L1 e Cache L2 sono componenti del tutto 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.

3.2 CACHE DI PRIMO E DI 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, il *flush* completo delle cache dopo un cambio di contesto a seguito di un evento di *page fault*.

Una singola linea di cache di primo o secondo livello si presenta dunque come in tabella 1 :

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

Tabella 1: Singola linea di cache L1/L2

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 analogamente ai sistemi Linux, dove i *PID* dei processi sono compresi tra 0 e 32768 ossia rappresentabili come intero senza segno a 16 bit. Si noti come tale architettura non è invece adatta a

sistemi Windows® in quanto i PID su tale sistema sono multipli 4 di e di tipo *DWORD* sebbene la compatibilità si possa semplicemente ottenere portando a 32 i bit riservati all'*ASID*.

3.3 STRUTTURE HARDWARE DI SUPPORTO ALL'ARCHITETTURA 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*.

3.3.1 *TLB*

Al fine di velocizzare quanto più possibile l'operazione di traduzione la CMM è dotata di un TLB che associa alla coppia $\langle Asid, VA \rangle$ l'indirizzo fisico del dato richiesto.

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

Tabella 2: Singola entry del TLB

La tabella 2 mostra una generica entrata del TLB : si noti come la struttura è pressochè identica a quella di una linea di cache con l'aggiunta dei bit di validità, di presenza e di modifica e dove ovviamente al posto del campo dati si trova un indirizzo fisico di memoria.

Una miglioria apportata al TLB da questo studio e che non era stata presa in considerazione precedentemente da [5] è la sua espansione da un piccolo numero di entrate e da un'associatività *fully* 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 energia consumata e area occupata.

Studi precedenti in [5] 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 memorie principali della passata generazione, è davvero necessario trasferire dati pari alla dimensione di una pagina nel momento in cui si verifica un evento di *page fault*? La risposta è stata 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 bit dei sistemi tradizionali ad un campo più piccolo a 20 bit che altro 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 (ovvero il numero di pagine fisiche da 32KB in una memoria principale di 8GB) fino al numero di bit necessari a rappresentarne 1048576 (considerando una memoria principale di 32GB) ossia è sicuramente minore o al più uguale a $log_2 1048576 = 20$.

Questo riduce del 29% la grandezza di una singola linea di TLB con conseguente risparmio di energia e area occupata.

3.3.2 *Bitmap*

A seguito di una richiesta di un dato non ancora presente in memoria si verifica quindi un *page fault* che causerà un trasferimento in memoria fisica della sola sottopagina relativa al dato richiesto e memorizzerà nel TLB un'entrata relativa alla pagina 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 come mostrato in tabella 3:

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

Tabella 3: Singola entry del TLB comprendente la *bitmap*

3.3.3 *SRAM*

Per realizzare l'astrazione di una memoria principale con indirizzamento *cache-like* viene utilizzata una memoria SRAM che agisce da *smistatore* degli indirizzi virtuali e partecipa così alla traduzione per ottenere l'indirizzo fisico.

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 4GB.

L'architettura CMM invece fa intervenire, nel caso di una TLB-miss, il meccanismo *cache-like* di traduzione dell'indirizzo virtuale che grazie anche all'indicizzazione a insieme velocizza sensibilmente l'operazione di traduzione. Prima di passare alla descrizione dell'architettura interna della cache della CMM è necessario esporre da quali campi sono costituiti gli indirizzi virtuali e fisici relativi alla CMM. La struttura di un indirizzo virtuale è presentata in tabella 4 :

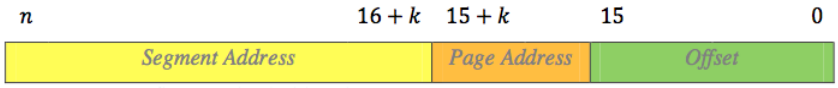

Tabella 4: Struttura di un indirizzo virtuale nell'architettura CMM

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

Ulteriore decisione progettuale che non era stata prevista in [5] è stata quella di dividere la memoria virtuale in segmenti ognuno delle quali capaci di contenere k pagine distinte. Ovviamente risulta che il numero di segmenti è pertanto 2^{n-15-k} , dove n è lunghezza totale dell'indirizzo virtuale di memoria.

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 $n = 64$ bit a m bit dove il valore di 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 modo 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 in [5] e ripetuta in quanto non pertinente con lo studio desiderato dall'autore di questo documento e in[10] e [14] è stata quella di presuppore un agente esterno, eventualmente software, che si occupi della veloce traduzione mediante tradizionale page walking dell'indirizzo di segmento.

Si noti come tale traduzione anche se effettuata via software non richiede valori di tempo elevati in quanto il valore di output è solo di $12 \div 14$ bit e pertanto può essere ottenuto mediante un *page walking* che può essere semplicemente implementato tramite una semplice *lookup table*.

Rimane quindi palese 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 in [5] per effettuare tale traduzione era quella di creare, all'interno della CMM, una struttura dati hardware chiamata *CMM TAG/Location* (realizzata trmite la memoria SRAM sopra citata) la cui struttura è riportata in tabella 5 :

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

Tabella 5: Struttura *TAG/Location*

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 è stata quella di introdurre una corrispondenza diretta 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 8 :

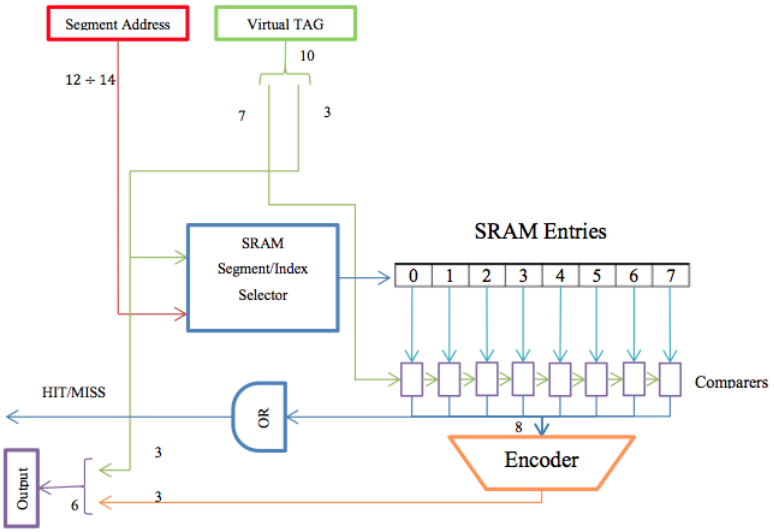

Figura 8: Struttura interna della SRAM

L'elenco dei passaggi effettuati durante la traduzione di un indirizzo virtuale 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; si è usato 8 poiché gli studi precedenti avevano dimostrato che 8−*way* era la migliore associatività possibile in un range da *direct-mapped* a 32-*way*) e mandare gli output 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 8GB fino a 35 bit per la versione a 32GB.

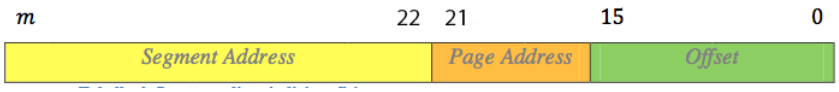

Tabella 6: Struttura di un indirizzo fisico

3.3.4 *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 in tabella 7 :

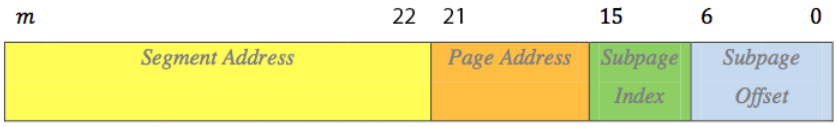

Tabella 7: Struttura definitiva di un indirizzo fisico

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 header 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 dispositivo FRTD e posizionarla nella corretta locazione in memoria 3D DRAM oltre a dover aggiornare tutte le strutture sopra citati che tengono traccia di presenza e/o modifica delle sottopagine, ovvero il TLB (nel caso di sub-*page fault* si deve solo modificare la *entry* corretta nel TLB, in caso di *page fault* si deve inserire una nuova *entry* nel TLB) . Nel caso di trasferimento da FRTD, causa la maggiore velocità di risposta che si associa a quest'ultimo se paragonato agli 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.

3.3.5 *Row buffers*

Per quanto riguarda l'accesso alla 3D DRAM stessa inoltre, 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 4 notiamo come

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 (come è facilmente visibile nella figura) e fanno quindi riferimento, nell layer 1, ad una singola porzione di logica statica di controllo nella quale sono contenuti i *row buffer*.

Organizzati come piccole cache tali *row buffer* velocizzano enormemente le operazioni di ottenimento del dato se questi era stato precedentemente acceduto, come già studiato in [3].

Generalmente ad ogni *bank* è associato un solo *row buffer*, tutta via nel nostor modello abbiamo voluto sperimentare l'aggiunta di un numero maggiore di *row buffer* per ogni *bank* in modo da aumentare l'*hit rate* tra il dato cercato in memoria e quello già presente nell'array di *row buffer*.

Il numero di *row buffer* supposti per ogni gruppo di banchi organizzati su più layer in un singolo rango è 4.

3.3.6 *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*.

È quindi ora possibile presentare un diagramma di flusso, in figura 9, che rappresenta tutte le operazioni effettuate dalla CMM dal momento di arrivo della richiesta al soddisfacimento della stessa.

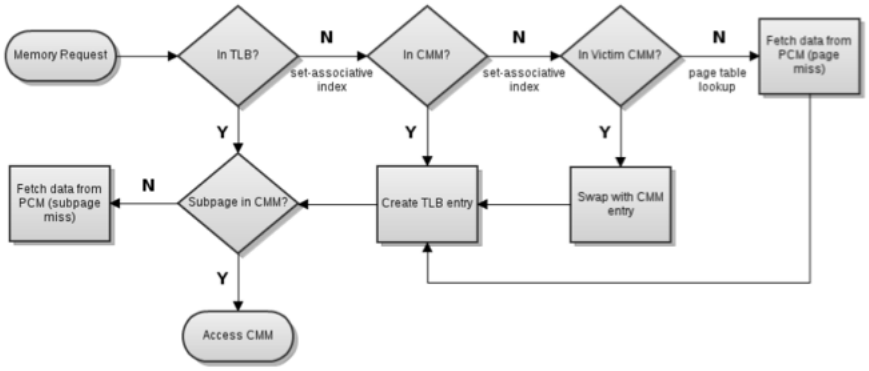

Figura 9: Algoritmo completo di accesso alla CMM

3.3.7 *Victim Cache*

La figura 9 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* il cui scopo è contenere le pagine selezionate come *vittime* e rimosse dalla memoria principale. In questo modo si evita la scrittura in FRTD della pagina di memoria

in modo tale che possa essere resa immediatamente disponibile nel caso in cui venisse referenziata poco dopo la sua rimozione.

L'uso di una *Victim Cache*, delle quali sono presenti, sotto un punto di vista generale, molte informazioni in letteratura, in CMM infatti non portava alcun beneficio visibile e dunque è stata disattivata in tutte le simulazioni.

3.3.8 *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 (*Flash Replacement Technology drive*), 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 ordini di grandezza più veloce, sia in lettura che in scrittura, rispetto ad un classico disco a rotazione magnetica: questa caratteristica è fondamentale per giustificare le scelte progettuale sopra presentate e per garantire le performance 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 pari a :

$$\#cycles = accessTime * cpuFrequency$$

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

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

porta ad una sensibile perdita di prestazioni in quanto 6000000 è un numero di cicli di clock notevolmente alto 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 non trascurabile questo ammontare di cicli di clock è effettivamente un guadagno se si pensa a quanti cicli si spendono per eseguire *context switch* in caso di *I/O* da disco a causa di un *page fault*.

4 PCM COME MAIN MEMORY E 3D-DRAM COME LLC

4.1 PCM COME MAIN MEMORY

Il sempre crescente numero di processori presenti nei moderni sistemi ad alte prestazioni richiede un sottosistema di memoria altamente scalabile che sia in grado di gestire *working set* applicatii sempre più esigenti. 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 da imputarsi alla memoria principale. Inoltre, come già detto, lo *scaling* della memoria è diventato un serio problema da affrontare; questa combinazione di fattori conduce all'inevitabile conseguenza che la tecnologia di base delle memorie 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.

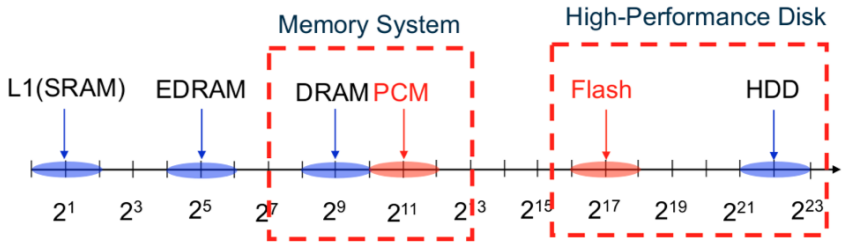

Figura 10: Latenze di accesso tipiche per varie tecnologie (in cicli di clock), da [12]

La figura 10 mostra latenze tipiche d'accesso per varie tecnologie di memoria. 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, ma la latenza che caratterizza

questo tipo di memorie e la loro scarsa durabilità nel tempo non le rende una valida alternativa scalabile per le memorie principali.

Le latenze più vicine a quelle di una DRAM rendono dunque la PCM un'attraente tecnologia per incrementare la capacità della memoria principale manentendo sensibilmente limitate le latenze di accesso. Si consideri inoltre che le celle PCM hanno una durata media 1000 o più volte superiore a quella di celle Flash.

4.1.1 *Organizzazione di una memoria principale realizzata tramite PCM*

In modo del tutto simile all'organizzazione di una memoria DRAM convenzionale, le celle PCM possono essere organizzate in *ranks* e *ban*ks. E' comunque possibile sfruttare le caratteristiche tipiche della PCM per rivedere ed ottimizzare l' architettura 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 dato nei *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 fessibilità nell'organizzazione dei *row buffer*s, 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, la quale è definita dal numero totale di bit nelle *bit-cells*. La larghezza del buffer è infatti un parametro critico di progettazione: essa determina il numero dei *sense amplifiers*, i quali sono gli elementi che maggiormente penalizzano il dispositivo di memoria, in termini di consumo energetico e di area.

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

4.2 PCM COME STORAGE SYSTEM

Al giorno d'oggi le memorie Flash in tecnologia NAND sono utilizzate nei *Solid-State Disks* (SSD). A differenze dei classici *Hard Disk Drive* (HDD), i quali si basano su componenti elettromeccaniche, 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 scrit-

ture 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 Tranlation 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 retrocompatibilità con le interfacce attuali e i sistemi operativi odierni.

4.3 PRESENTE E FUTURO DELLE PCM

In conclusione PCM sono una nuova ed emergente tecnologia per quanto riguarda le memoria: attualmente molto utilizzate nei sistemi embedded 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, possiedono ancora una lunga strada dello sviluppo innanzi a loro poiché ancora poco utilizzate e quindi relativamente poco studiate e inoltre 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; onde inoltre tutelarsi da un'eventuale non previsto sottosviluppo si è cercato di affianca-

re alla "lenta" PCM un dispositivo di archiviazione temporaneo più veloce, la memoria DRAM con approccio tridimensionale.

4.4 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 (a cui ci riferiremo di seguito con l'acronimo L3 oppure LLC), molto più grande delle precedenti e condivisa tra tutti i core. Le dimensioni di tale cache sono generalmente nell'ordine di $4MB - 12MB$ e sono affiancate dalla memoria RAM principale.

Nei sistemi in studio in questo documento tuttavia ciò che affianca la L3 risulta essere una memoria principale basata su PCM: è ovvio quindi che, confrontando le dimensioni piccole, nell'ordine della decina di MB, con le enormi dimensioni di una PCM e soprattutto considerando il *gap* di velocità tra le due tecnologie, il solo utilizzo della tecnologia PCM non garantisce 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 3D-LLC tuttavia disaccoppia la bivalenza tra cache e memoria principale adottata dall'architettura CMM e mantiene distinte le due entità.

4.4.1 *Coerenza della 3D-DRAM come 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-Cach*e privata di primo livello e un'altresì privata cache di secondo livello. Lo schema semplificato di questo tipo di architettura è proposto in figura 11 :

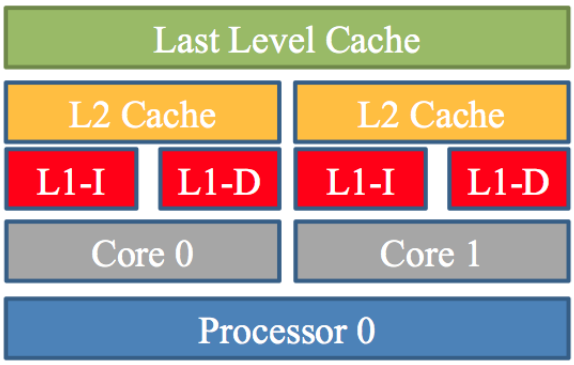

Figura 11: Architettura semplificata con LLC

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 privata 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 4.*c* 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 , in[9], 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.

4.5 CARATTERISTICHE 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.

4.5.1 *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 del processore come facilmente visibile in figura 12 :

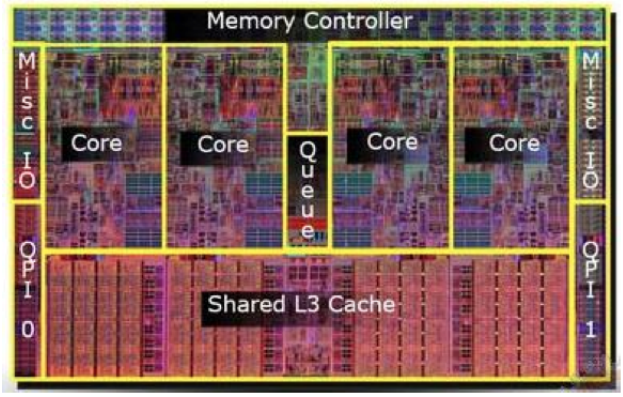

Figura 12: Architettura di un processore i7 di prima generazione (Nehalem)

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 è descritta in tabella 8 :

Physical Tag ≤ 49 bit

Tabella 8: Struttura di una linea di cache nella SRAM della 3DLLC

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 principale.

4.5.2 *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 questo tipo di memoria risulta dunque quello riportato in figura 13 :

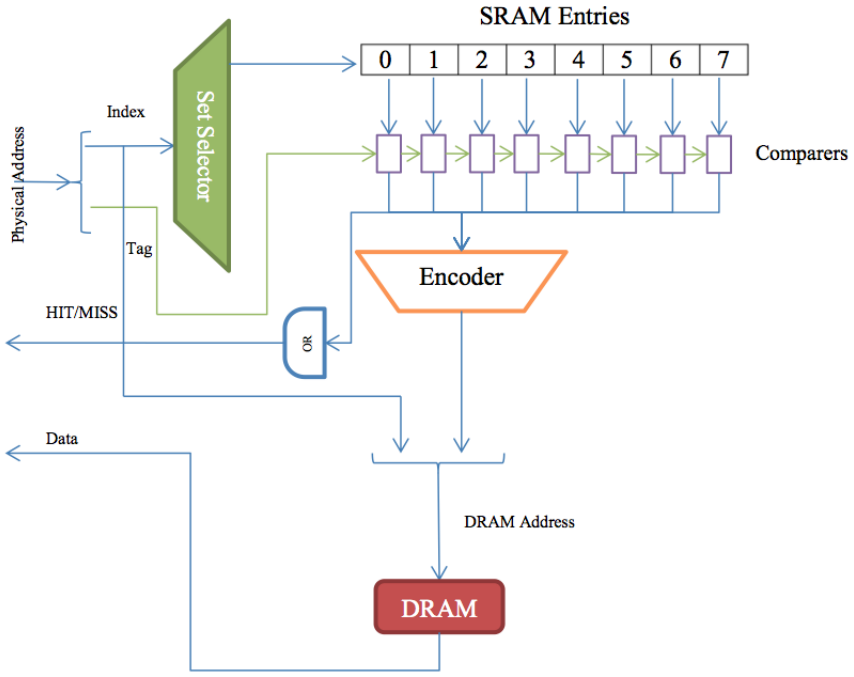

Figura 13: Struttra del sistema 3D-DRAM as LLC

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.

4.5.3 *Row-Buffers*

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

4.6 CARATTERISTICHE DELLA PCM AS MAIN MEMORY

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).

I primi prodotti di questo tipo realizzati e commercializzati da Micron[1] sono internamente strutturati e quindi esternamente visibili come delle classiche memorie RAM, da [12]. Questo permette agli at-

1 http://www.micron.com/products/phase-change-memory/parallel-pcm - Micron Technology Inc.

tuali 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.

4.6.1 *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 LLC 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 8GB (la più grande dimensione considerata nelle simulazioni) si avrebbe una SRAM, da realizzare tramite più costosa tecnologia CMOS, la cui grandezza è data da:

$$\#SRAM_{Entries} = \frac{DRAM_{size}}{SUBPAGE_{size}}$$

e nel caso in studio avremmo quindi un numero di entrate pari a :

$$\frac{8 \cdot 2^{30} bytes}{128 \frac{bytes}{entries}} = 67108864 Entries = 64 M Entries$$

Dato che :

$$SRAM_{size} = \#SRAM_{entries} \cdot ENTRY_{size}$$

si otterrebbe una dimensione totale della SRAM pari a :

$$64 M Entries \cdot \left\lceil \frac{49 bit}{8 \frac{bit}{byte}} \right\rceil = 448 MB$$

Considerando invece la dimensione della cache line pari a 1024 byte si avrebbe un numero di entry nella SRAM pari a :

$$\#SRAM_{Entries} = 8MEntries$$

per una dimensione totale della SRAM pari a :

$$SRAM_{size} = 56MB$$

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 $\frac{1024}{128} = 8$).

4.6.2 *Rowbuffers*

Come si è già menzionato in precedenza la PCM utilizzata come memoria principale è internamente strutturata come una normale ed attuale memoria RAM. Questo ovviamente significa che sono presenti anche i *row buffer* 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:

1. Row Buffer 3D-DRAM;
2. 3D-DRAM;
3. Row Buffer PCM;
4. PCM

Un'importante considerazione da tenere conto nei row buffer 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 *shutdown* del sistema o finché un ulteriore dato dallo stesso banco verrà richiesto.

Miglioria aggiuntiva proposta in [12] è stata quella di disaccoppiare i *sense amplifier* dai row buffer veri e propri in maniera tale da poter realizzare i medesimi multiplexando i vari dati provenienti dai diversi sense amplifier 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 *bitcell* pertinenti.

4.7 PANORAMICA RIASSUNTIVA DELL'ARCHITETTURA 3D-DRAM AS LLC E PCM AS MAIN MEMORY

Una visione a blocchi dell'intero sistema 3D-DRAM-LLC + PCM-MM è visibile in figura 14 :

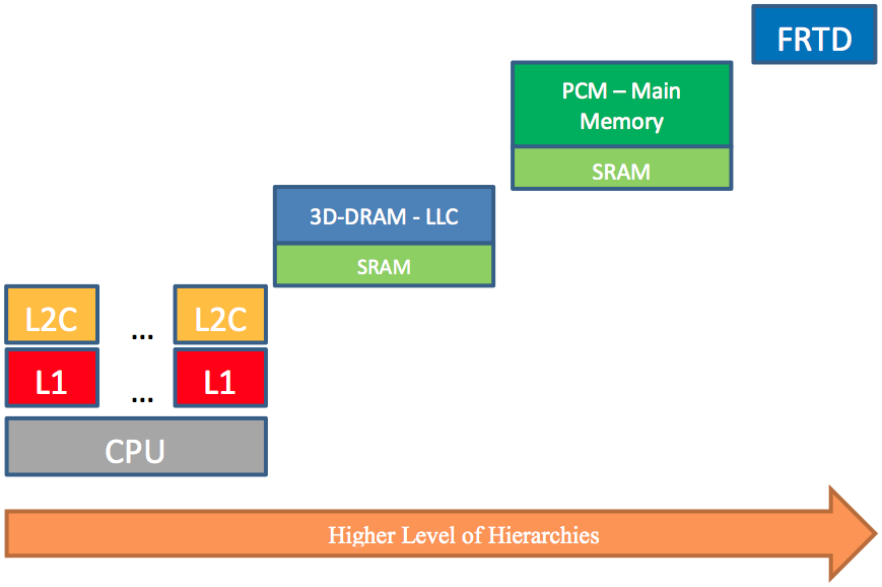

Figura 14: Visione di insieme del sistema 3D-DRAM-LLC + PCM-MM

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 swap space. Per ulteriori informazioni sulle FRTD si veda il sottoparagrafo ad esse relativo riportato nella sezione della CMM.

5

PHASE CHANGE MEMORY COME MEMORIA PRINCIPALE E 3D-DRAM COME PARTE DELLA MEMORIA PRINCIPALE

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 principale possa viariare in relazione alla presenza o meno delle pagine fisiche in dispositivi di memoria caratterizzati da latenze e consumi ben differenti tra di loro.

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 non è banale e richiede una serie di considerazione 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 addizionale al classico controllore di memoria in modo tale che possa gestire le operazioni di memoria traducendo gli indirizzi fisici (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 15 è presentato un semplice schema che riassume quanto appena detto.

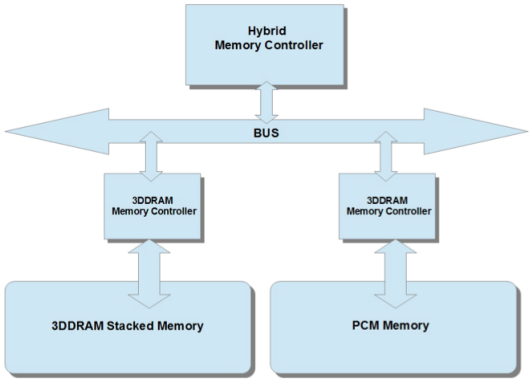

Figura 15: Architettura ibrida di memoria 3D-DRAM e PCM

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

5.1 POLITICHE DI SMISTAMENTO HARDWARE-BASED

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 di 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 16 è riportato un semplice schema che mostra l'organizzazione delle pagine di memoria e il loro smistamento casuale.

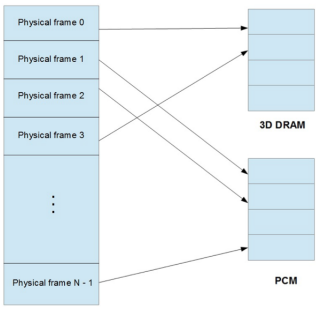

Figura 16: Architettura ibrida di memoria 3D-DRAM e PCM

5.1.1 *Politica di smistamento a gruppi*

Una politica più intelligente potrebbe essere quella in cui le pagine risultino organizzate in sottinsiemi, 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 suppona ad esempio che il rapporto tra la dimensione della 3D DRAM e la PCM sia :

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

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

$$\#groups = \frac{3DDRAM_{size_{in_{GB}}} + PCM_{size_{in_{GB}}}}{(a+b) \cdot size_{of page}}$$

A questo 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 17 :

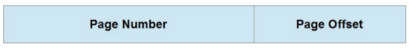

Figura 17: Indirizzo fisico in uscita dal sistema CMM

Si immagini una memoria composta da 2 GB di 3D DRAM e 8 GB di PCM: in tal caso il rapporto tra le due memorie sarà $\frac{2}{8}$ e si otterranno quindi gruppi di 10 pagine fisiche in cui le prime due risiederanno nella 3D DRAM e le rimanenti otto nella memoria PCM.

Dividendo a questo punto il campo *Page Number* in figura per il numero di pagine in un gruppo, si ottiene l'indice del gruppo di pagine all'interno della memoria unificata che abbiamo chiamato *groupNumber*; effettuando poi un'operazione di modulo tra *Page Number* e il numero di pagine in un gruppo si ottiene il numero di pagina all'interno del gruppo stesso che noi chiameremo *groupOffset*.

Nel caso in cui il valore di *groupOffset* fosse minore del numero di pagine di 3D DRAM presenti in ciascun gruppo allora il nostro dato ricadrà in 3D DRAM e il suo indirizzo fisico ibrido di memoria sarà:

$$3DDRAM_{physical\,Address} = groupNumber * PAGE_{size} + groupOffset$$

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

$$PCM_{physical\,Address} = groupNumber * PAGE_{size} + (groupOffset - \#3DDRAM_{pages})$$

Il meccanismo finora indicato viene riportato schematicamente in figura 18 al fine di fornirne una più immediata e semplice comprensione al lettore :

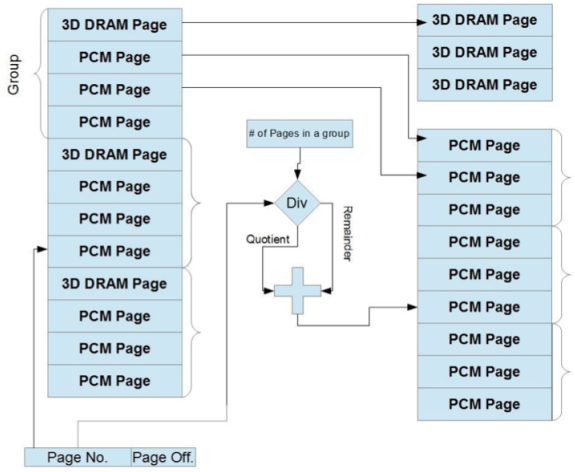

Figura 18: Algoritmo di smistamento dei dati nella memoria ibrida. Da notare come, per semplicità, è stata omessa in figura la sottrazione dal resto della divisione intera del numero di pagine occupate dalla 3D DRAM nel singolo gruppo

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.

5.1.2 *Politica di smistamento a gruppi 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 :

$$log_2\alpha \in \mathbb{N}$$

dovremo allora considerare b bit subito superiore al *PageOffset*, dove $b = log_2\alpha$; avremo 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 superiore al *PageOffset*, dove :

$$c_{3D} = log_2(\#Pages_{3D_{group}})$$

se l'analisi dei b bit indica che la pagina risiede in 3D DRAM, altrimenti :

$$c_{3D} = log_2(\#Pages_{PCM_{group}})$$

In figura 19 è presentato un esempio esplicativo dell'algoritmo sopra presentato.

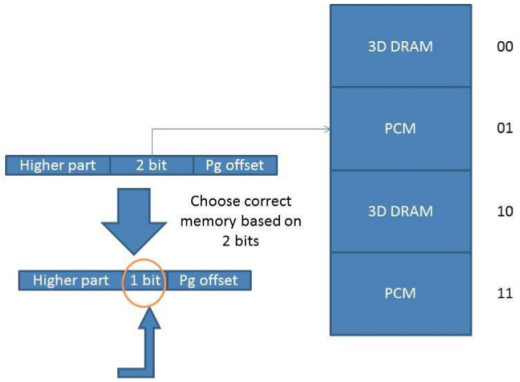

Figura 19: Politica Random Mixed alternative

In questo tipo di politicadi 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 stato nominato : **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 recessarie.

5.1.3 *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 finchè una delle due , la più piccola ovviamente, non termina, dopodichè non ci

sarà più alternanza e saranno presenti solamente le pagine relative al dispositivo di memoria più grande. Un esempio chiarificativo di questo approccio è presentato in figura 20 :

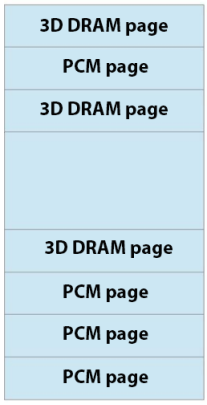

Figura 20: Politica di smistamento hard

Questo tipo di politica risulta implementabile con semplici operazioni hardware, poichè 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 psedo-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 bits) :

Algoritmo 5.1 Algoritmo per il calcolo dell'indirizzo fisico ibrido di memoria nella politica di smistamento forte

```
//check the parity of the 16th bit
if((PhysicalAddress && 0x00...08000) == 0x00...00000)
{
        //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)
}
```

5.1.4 *Politica di smistamento contiguo*

Un'altra politica di smistamento studiata è quella denominata : **politica di smistamento contiguo**.

Un'altra possibile politica di smistamento nella memoria ibrida può essere quella 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.

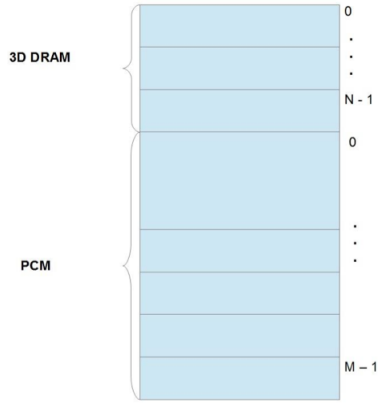

Figura 21: Politica di smistamento Random Continuous

Tale libertà consente dunque agli sviluppatori di sistemi operativi ampi margini di manovra anche nel caso in cui tale soluzione possa presentarsi sul mercato prima ancora che studi più approfonditi sull'organizzazione del kernel riescano a sfruttare al meglio una tale organizzazione, vedi figura 21.

5.1.5 *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'obiettivo 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 tenere traccia degli accessi a ciascuna della pagine fisiche e portare dinamicamente le più accedute nella memoria più veloce. Per ogni pagina bisognerebbe dunque memorizzare un contatore almeno grande

un byte e l'indirizzo in una delle due memorie in cui la pagina fisica è stata realmente memorizzata, per un totale di $1 + 6 = 7$ bytes a pagina. Questo comporta l'utilizzo di una struttura hardware che memorizza $7 \cdot 327680 = 2293760$ byte, i quali corrispondono a $\sim$ 2MB: 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 svantaggiare 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 idle del bus.

5.2 POLITICHE KERNEL-BASED

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

Si potrebbe pensare di fornire agli sviluppatori di applicazioni un sottoinsieme di *kernel APIs* per poter specificare dove voler salvare il proprio spazio di memoria.

Si potrebbe 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.

Parte III

STRUMENTAZIONI DI SIMULAZIONE

Nella Parte I verranno analizzati gli strumenti di modellazione e simulazione utilizzati in questo lavoro : *HP CACTI*, *CACTI-3DD* e *Wind River SIMICS*. Si cercherà di far comprendere la loro utilità, le finalità e le modalità con cui sono stati utilizzati : parametri di input, file di configurazione e componenti di output. Un analisi più approfondita dei valori effettivamente utilizzati per la configurazione di questi strumenti sarà trattata nella Parte relativa alle simulazioni e alla analisi dei risultati.

6 SIMULATORE HP CACTI

CACTI 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 che si vuole studiare, quali :

- tempi di accesso alla memoria
- tempo di cycle
- area
- potenza di dispersione
- potenza dinamica

attraverso le quali il progettista può far fronte agli svariati trade-off derivanti dalle scelte effettuate sui parametri di progettazione[1].

Man mano che i processi produttivi si andavano evolvendo, CACTI ha subito pesanti modifiche [2] 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 utlima generazione quali le *3D Stacked DRAM* o le *Phase Change Memories.*

CACTI mette a disposizione inoltre una interfaccia web con la quale 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.

CACTI Web interface : http://quid.hpl.hp.com:9081/cacti/

6.1 PARAMETRI DI CONFIGURAZIONE DI CACTI

Prenderemo 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 che si vuole simulare; in particolare la versione di CACTI

1 Total amount of memory, bus lenght, cell type and technology, block size, etc.

2 Attualmente la versione più aggiornata è CACTI 6.5, le cui specifiche sono reperibili a questo indirizzo http://www.hpl.hp.com/research/cacti/

che trattiamo in questo capitolo permette la simulazione di 3 tipi di memoria :

- **Cache**, comprensiva quindi di un *tag array*, selezionabile tramite la parola chiave **cam** (*content-addressable memory)*
- **Ram**, modello simile ad un *Register File*, selezionabile tramite la parola chiave **ram**
- **Main memory**, memoria senza tag array cui ogni accesso avviene alla granularità di una pagina di memoria, selezionabile tramite le parole chiavi **main memory**

Per i nostri studi faremo riferimento solamente 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.

6.1.1 *Parametri di configurazione per entrambi tipi di memoria*

```
-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 GBytes della memoria

```
-block size <blocksize>
```

- Per la modellazione di memorie cache questo parametro specifica la dimensione in Bytes della parte dati all'interno di uan 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, etc).* Per la nostra trattazione considereremo solamente tre valori :
  - *itrs-hp : International Technology Roadmap for Semiconductors - High Performance* - Processo ad alte prestazione, tipico delle memorie cache
  - *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
  - *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[3], e non verranno trattati in questi capitoli.

6.2 PARAMETRI DI CONFIGURAZIONE PER 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 servono per 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 nel paragrafo6.1.1.

```
-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) - "normal"
```

- 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*)

3 Sono stati sempre considerati vincoli di modellazione piuttosto stringenti in modo tale da avere una simulazione il più conforme possibile a quanto speificato nei parametri ci configurazione

```
-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*
  - 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 μεμορια nel caso in cui si prevede il suo utilizzo in una architettura multiprocessore o non.
  - *NUCA* : *Non-Uniform Cache Access*
  - *UCA : Uniform Cache Access*

6.2.1 *Parametri di configurazione per 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.

6.3 FILE 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.

6.3.1 *Componenti di output per 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.

I 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;
4. Tempo ed energia per lo scaricamento/caricamento dal *bit-array* del dato cercato;
5. Altre operazione di minore importanza non essenziali al fine della trattazione;

`Total leakage power of a bank (mW)`

- Potenza di dispersione di 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.

6.3.2 *Componenti di output per memorie principali*

```
t_RCD (ns)
```

- Row to Column command delay, il tempo che il dato impiega ad arrivare dalle *bitlines* 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 necessario per precaricare 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*[4] 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

Operazioni di lettura e scrittura in DRAM

I parametri sopra elencati devono essere oculatamenti 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-buffer* 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.

- ***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 *activated* dalla operazione di lettura/scrittura precedente a quella presa in esame.

4 *Durante l'operazione di precharge la bitline viene caricata ad un livello intermedio tra il valore di soglia '0' e il valore di soglia '1', in modo tale che alla selezione della wordline i sense amplifiers sottostanti collegati alle bitline potranno registrare il valore salvato nella cella di memoria; per questo motivo ogni operazione di precharge risulta essere distruttiva, ovvero ad ogni accesso lo scaricamento della wordline nei rowbuffer comporta la distruzione del dato stesso, che dovrà, ad operazione (lettura/scrittura) terminata, essere riscritto.*

  - Delay di operazione di lettura/scrittura : *t_CAS*
  - Energia consumata dalla operazione di lettura/scrittura : *Read_Energy/Write_Energy*

- ***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 poichè non interessato dall'operazione di memoria precedente a quella presa in esame.
  - Delay di operazione di lettura/scrittura : *t_CAS* + *t_RAS*
  - Energia consumata dalla operazione di lettura/scrittura : *Read_Energy/Write_Energy* + *Activation_Energy*

- ***SEMI-MISS*** - Si verifica quando il banco di memoria contenente il dato da dover prelevare per l'operazione di lettura/scrittura risulta già *activated* poichè utilizzato dall'operazione di memoria precedente, ma tale dato non risulta essere nel *row buffer* e deve essere scaricato dal *bit-array.*
  - Delay di operazione di lettura/scrittura : *t_RP* + *t_RCS* + *t_CAS*
  - Energia consumata dalla operazione di lettura/scrittura : *Read_Energy/Write_Energy* + *Precharge_Energy*

- ***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.*
  - Delay di operazione di lettura/scrittura : *t_RP* + *t_RCS* + *t_CAS* + *t_RAS*
  - Energia consumata dalla operazione di lettura/scrittura : *Read_Energy/Write_Energy* + *Activation_Energy* + *Precharge_Energy*

7

CACTI-3DD

CACTI-3DD è una recente evoluzione di CACTI volta alla modellazione di memorie 3D-DRAM Stacked. Come si legge in [2] questa versione del potente strumento di modellazione di memorie rende possibile lo studio approfondito di questa recente e innovativa tecnologia, già ampiamente presentata nel capitolo XXX, e restituisce informazioni dettagliate inerenti ogni operazione di memoria, lettura e scrittura, energie consumate durante le operazioni e dal bus interno costituito dalle *Through Silicon Vias*.

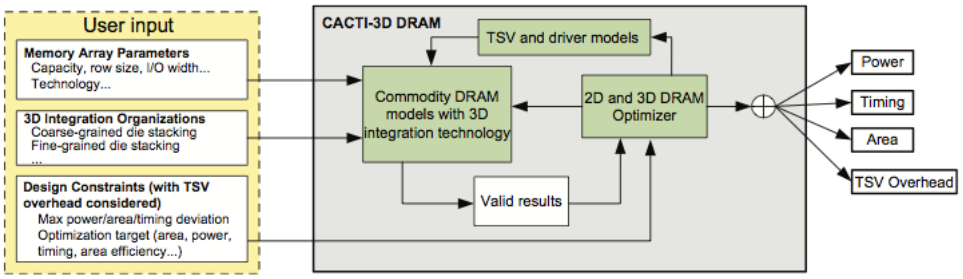

Figura 22: Diagramma a blocchi del framework di CACTI-3DD

In figura 22 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 *Through Silicon Vias*.

Il software, oltre a tutti i parametri già ampiamente argomentati nei capitoli 6.1.1 e 6.2.1 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 desiderano 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 da [7] e già ampiamente argomentate nel paragrafo 1.2.2 di cui si riporta il modello in figura 23.

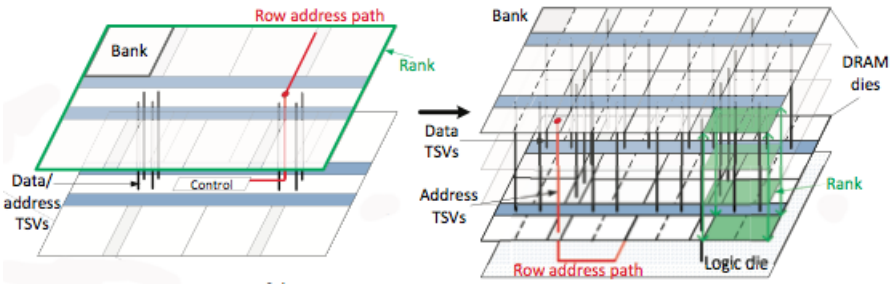

Figura 23: Modello architetturale per *WIDE-3D* e *TRUE-3D Stacked DRAM*, da [2]

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 *Through Silicon Vias* che collegano banchi di *die stacked* differenti e appartenti allo stesso *rank* 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 songoli banchi *stacked*, ulteriore ottimizzazione del paradigma *TRUE-3D*, continuando ad utilizzare il classico modello *planar-bank*.

Il modello del *bank-stacked die*, da [7], è riportato in figura 24.

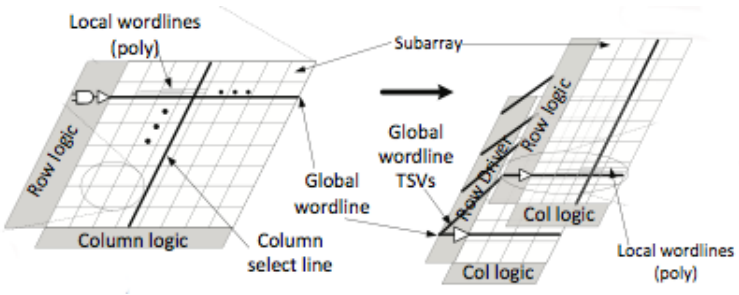

Figura 24: Evoluzione dal *planar-bank* allo *Stacked-bank*, da [2]

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 *rowbuffers*. 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

rowbuffers[1] possa avvenire in parallelo. Notare che anche la comunicazione tra i vari *sub-bank*s avviene per mezzo delle *Through Silicon Vias*.

1 *Ovviamente in questa configurazione sarà presente un rowbuffer di dimensioni ridotte, uno per ogni sub-bank, rispetto all'approccio classico planar-bank in cui memorizzare parte del dato presente nella bit-array. Il dato dovrà poi essere opportunamente ricostruito per essere inoltrato all'unità di elaborazione centrale.*

8 WIND RIVER SIMICS

Wind River SIMICS[1] è un *full system simulator* che mette a disposizione diversi strumenti per la modellazione di architetture *user-defined* da poter integrare, testare e debuggare in un ricco set di sistemi moderni preconfigurati nel sistema.

Con l'evolversi della tecnologia, i sistemi moderni sono diventati sempre più complessi : troviamo sistemi con processori *general purpose multi-core* in grado di supportare molteplici sistemi operativi, sistemi *HPC* con centinaia di *core* fisici e migliaia di *core* virtuali con enormi e complessi sistemi di memoria fisici o *cloud-based*. Di fronte 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 25 possiamo vedere uno schema ad alto livello del framework di simulazione offerto da SIMICS.

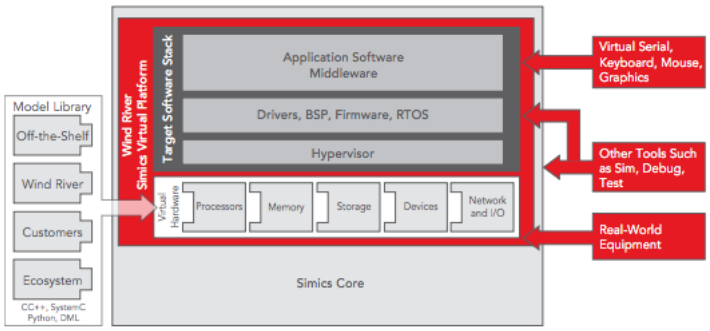

Figura 25: Framework di simulazione di Wind River SIMICS

Come si può vedere 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

1 http://www.windriver.com/products/simics/

- AMD
- Intel
- MIPS
- ARM
- 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[2]

, device di rete, supporti rimovibili quali CD-ROM, DVD-ROM, etc. . Ma quello che più interessa i progettisti è la possibilità di utilizzare moduli aggiuntivi, preconfigurati 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*.

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.

Il nostro lavoro è stato la definizione di una architettura di memoria attraverso la creazione di nuovi moduli, la creazione di un target, la installazione di un sistema operativo sopra di esso, la scelta di applicazioni utente da eseguire su questo sistema e la raccolta di statistiche inereti le performance ottenute dall'architettura definita.

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

2 In realtà questo risulta solamente essere un limite superiore virtuale per l'allocazione di memoria per le applicazioni e per il sistema, in quanto SIMICS non simula un unità di memoria principale ma utilizza la memoria del sistema host come propria memoria principale

1. Scelta di un target per il sistema (per le nostre simulazioni abbiamo usato 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 e esecuzione di programmi applicativi sulla macchina virtuale volti al testing dell'architettura che si vuole simulare (per lo più si tratta di programmi applicativi *benchmark*, e nel nostro caso sono stati utilizzati applicativi estratti dalle suite *SPEC2006* e *OLTP*);
5. Creazione di un *checkpoint*, ovvero una "istantanea" dell'intero sistema : contenuto della RAM, contenuto dell'Hard Disk, etc, 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[3];
7. Avvio del sistema simulato con il checkpoint sopra menzionato[4] 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.

3 Questo comporta ovviamente la scrittura di nuovi moduli per la simulazione di device nel caso in cui questi non siano disponibili *Off-the-Shelf* nella libreria moduli di SIMICS

4 E' possibile eseguire gli applicativi dentro al checkpoint per un numero di steps/istruzioni desiderato in modo da poter calcolare a posteriori informazioni quali potenza dissipata oppire istruzioni per ciclo

9 BENCHMARK SUITES

In questo capitolo verranno brevemente esaminate le suite di benchmark utilizzate per lo stressing test della architetture presentate nei capitoli 3, 4 e 5.

Le suite utilizzate sono per i test sono state due :

1. **SPEC CPU2006**[1] : 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 testing di CPU e sottosistemi di memoria;

   Standard Performance Evaluation Corporation

2. **OLTP**[2] : suite di applicativi, caratterizati da un elevato *footprint* di memoria, rivolti al test di architetture *Cloud* e HPC.

   On-Line Transaction Processing

9.1 SPEC CPU2006

SPEC CPU2006 è una suite di benchmark che mette a disposizione una vasta gamma di applicativi per il calcolo di parametri di valutazone 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 computazione tipico di applicazioni in ambito scentifico.

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. *Gli workload* disponibili sono 3, e sono riportati di seguito ordinati secondo la complessità computazione e la quantità di memoria utilizzata[3] :

1. *Test*
2. *Train*
3. *Referernce*

Di seguito viene riportato un sottoinsieme di questi benchmark utilizzati nella nostra simulazione e una loro breve descrizione.

401.bzip2

- **Area di applicazione** : compressione

1 http://www.spec.org/cpu2006/
2 http://oltpbenchmark.com/
3 Notare che variare il *workload* significa selezionare un diverso set di input per il benchmark

- **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 e 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** : inteligenza 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** : compressioe 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** : cromodinamica 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** : fluidonamica 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-sflzate*;

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** :
- **Descrizione** : il benchmark simula grandi sistemi biomolecolari.

9.2 OLTP

OLTP è una suite di *cloud* benchmarks rivolti al testing 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

- **Descrizione** : il benchmark simula caratteristiche di workload di un sito di aste e annunci web.

Epinions

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

TATP

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

SEATS

- **Descrizione** : il benchmark simula un sito di prenotazione di voli aerei, comprese le ricerche dei voli e la loro prenotazione.

Parte IV

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 si passa ad analizzare le simulazioni e i risultati ottenuti attraverso queste. Verrà brevemente analizzata la configurazione di sistema utilizzata per le nostre simulazioni con SIMICS (CPU, frequenza, sistema operativo, etc.) per poi passare in rassegna i risultati ottenuti, in termini di *IPC* (*Instructions Per Cycle*) e consumo di potenza, paragonando la nostra architettura ad una baseline costituita da un sottosistema di memoria principale molto generoso. Si noterà come la nostra architettura presenterà notevoli incrementi prestazionali (fino al 45% in termini di *IPC*) pur mantenendo paragonabili i consumi di potenza.

10 EXPERIMENTAL SETUP

Come già detto nei relativi capitoli, le nostre simulazioni si sono state effetuate 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 [5], 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, nonchè per la valutazione del consumo di potenza. La macchina target su cui sono state effettuate le simulazione è un processore $x86-64$ "Hammer" 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. SPEC CPU2006 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 sè, 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 SPEC CPU2006 sono i medesimi utilizzati in [5].

Per quanto riguarda i Cloud Benchmark, la seguente tabella illustrerà i diversi mix utilizzati. Da notare come il footprint di ciascun mix è nell'ordine dei $\sim$ 15 GB, anche se, facendo questi mix capo ad un server MySQL, raggiungono un footprint dell'ordine dei $\sim$ 20 GB.

Mix	Benchmark 1	Benchmark 2	Benchmark 3	Benchmark 4
AuctionMark	autctionmark	auctionmark	sjeng	stream
Epionions	epinions	epinions	sjeng	stream
TATP	tatp	tatp	sjeng	stream
Seats	seats	seats	sjeng	stream

Tabella 9: OLTP Benchmark

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 direttamene l'attività di gestione di richieste remote.

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 finchè 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.

11

SIMULAZIONI ED ANALISI DEI RISULTATI OTTENUTE CON CACTI

11.1 2D DRAM BASELINE

Come ogni buono studio è innanzitutto necessario 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, realizzata 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. 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.

Nonostante la fisica irrealizzabilità 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 in tabella 10 :

Nome Simbolico	**Grandezza (GB)**	**Numero di banchi**	**Page Size (KB)**
2D-DRAM_8GB_8Banks_4KBpP	8	8	4
2D-DRAM_8GB_16Banks_4KBpP	8	16	4
2D-DRAM_16GB_8Banks_4KBpP	16	8	4
2D-DRAM_16GB_16Banks_4KBpP	16	16	4

Tabella 10: Tipi di memoria analizzati per la determinazione della Baseline

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 infatti, hanno, 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 in figura 26 per quanto riguarda le i delay di accesso, in figura 27 per le componenti energetiche e in figura 28 per le componenti di area :

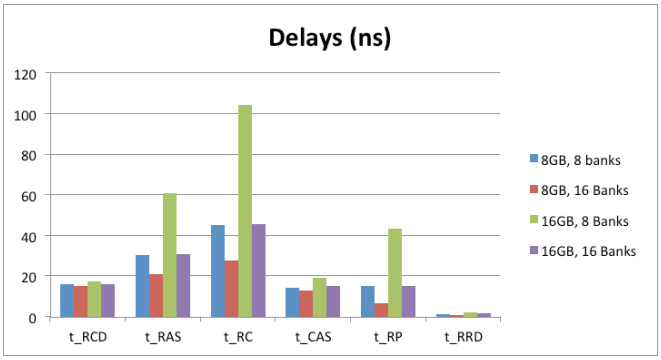

Figura 26: Delay (ns) per le componenti di accesso alla DRAM

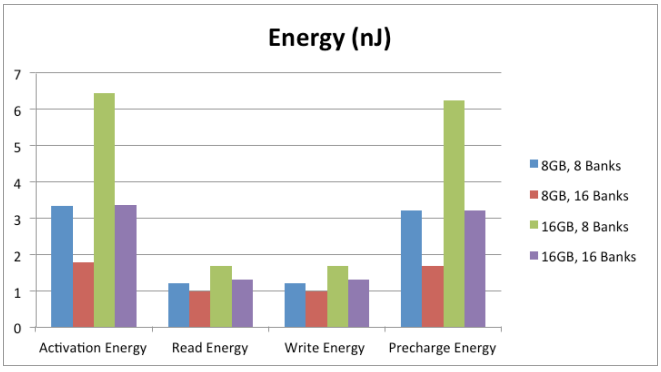

Figura 27: Energia (nJ) consumata per un singolo accesso in memoria

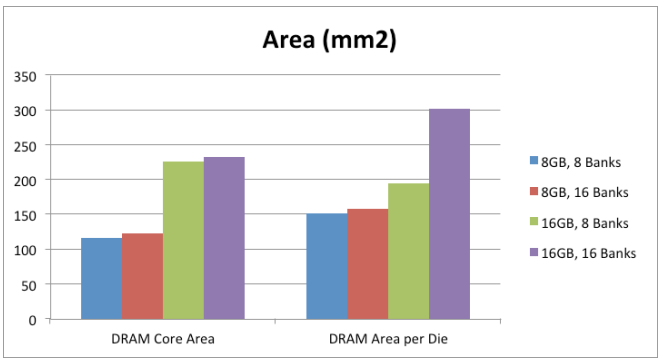

Figura 28: Area (mm^2) delle varie memorie

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 di 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 $20mm^2$.

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 10 è ***2D-DRAM_8GB_16Banks_4KBpP***.

11.2 TLB - TRANLSATION LOOKASIDE BUFFER

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 in [5] 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.

I TLB simulati con CACTI sono quelli riportati in tabella 11 :

Nome Simbolico	**Numero di 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

Tabella 11: Tipi di TLB simulati con Cacti

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

I grafici in figura 29, 30 e 31 mostrano le performance ottenute dai TLB appena esposti, rispettivamente per quanto riguarda delay di accesso, energia consumata per ogni accesso e area occupata dal *die*.

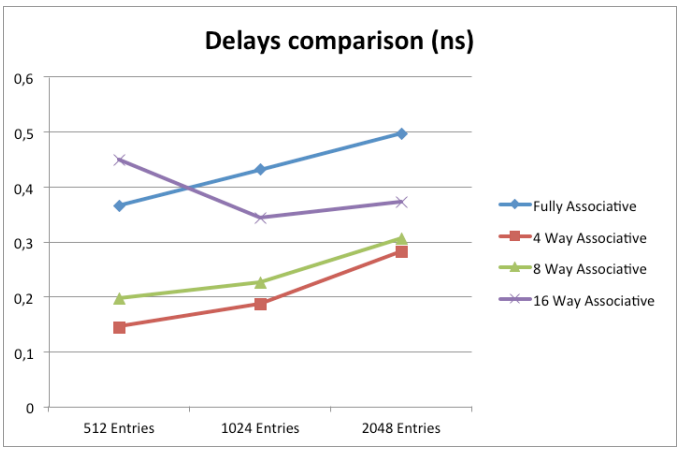

Figura 29: Confronto fra i tempi di accesso al TLB

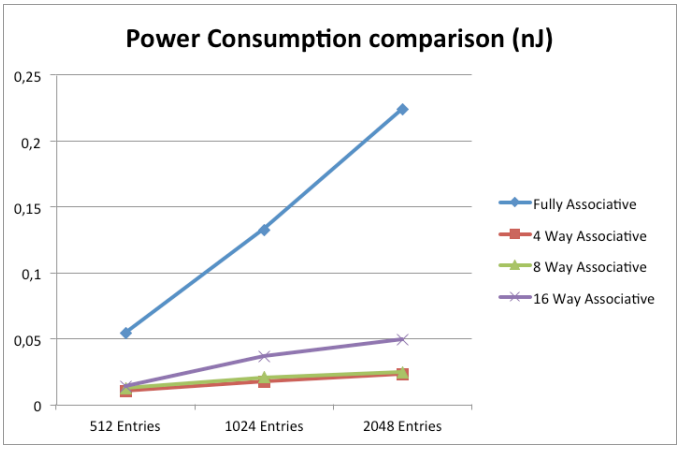

Figura 30: Confronto tra le le energia consumate dagli accessi in TLB

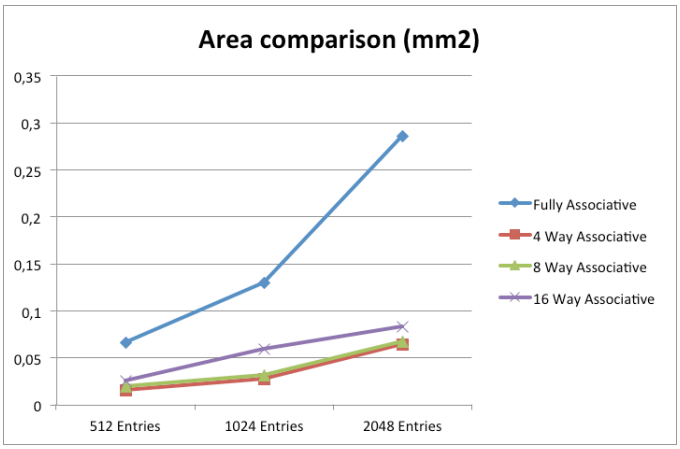

Figura 31: Cofronto tra le aree occupate dai TLB

Come è facile notare, 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 duqnue 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:

1. 1024 entrate, 4-way associative
2. 1024 entrate, 8-way associative
3. 2048 entrate, 4-way associative
4. 2048 entrate, 8-way associative

11.3 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 [5] 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:

$$\frac{(8.2^{30})}{2^{15}} = 8 \cdot 2^{15} = 262144$$

per la versione a 8 GB,

$$\frac{(16.2^{30})}{2^{15}} = 16 \cdot 2^{15} = 524888$$

per la versione a 16 GB e infine

$$\frac{(32.2^{30})}{2^{15}} = 32 \cdot 2^{15} = 1048576$$

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 in figura 32 :

Figura 32: Struttura finale di una entrata in SRAM

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

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

ad un massimo di :

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

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 12 mostra, in maniera non dissimile dalle altre, le differenti varianti di SRAM testate :

Nome Simbolico	**Numero di 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_1Bank	1048576	2
SRAM_Cache_IndexTrick_32GB_1Bank	1048576	4

Tabella 12: Lista delle varie configurazioni di SRAM testate

I risultati delle simulazioni sono riassunti nei grafici in figura x e y rispettivamente per quanto riguarda delay di accesso e energia consumata per ogni accesso.

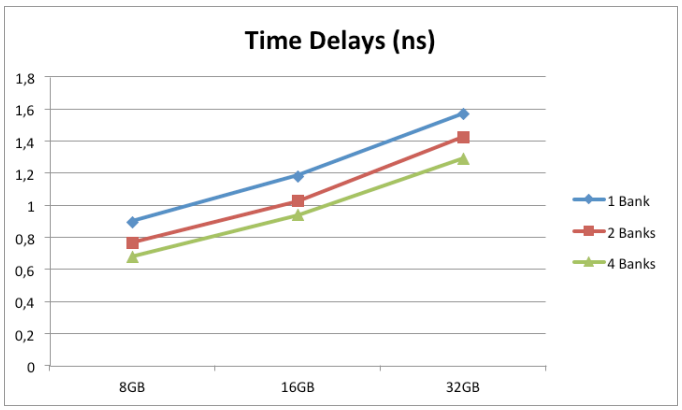

Figura 33: Latenze per le varie SRAM

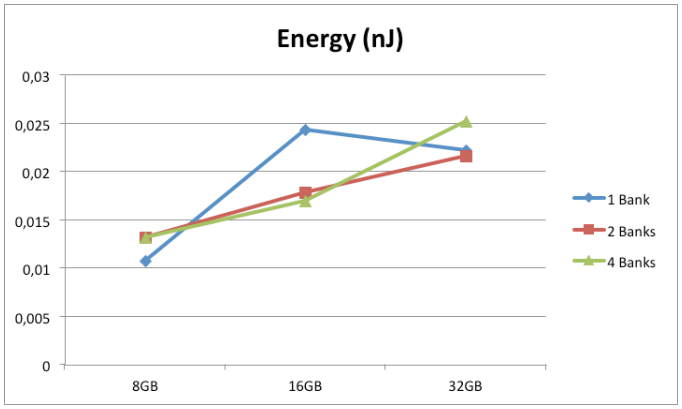

Figura 34: Energia consumata da un singolo accesso nelle varie SRAM

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 numerose volte affermato, 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.

11.4 3D DRAM

È ora il momento di analizzare la principale componente del sistema CMM: la 3D-DRAM. Numerose volte è già stato esposto quale sia l'insieme di dimensioni adottate per le soluzioni hardware ossia 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.

La tabella 13 mostra tutte le varianti analizzate :

Nome Simbolico	Grandezza (GB)	Numero di 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

Tabella 13: Varianti della 3D DRAM simulate in CACTI

Il numero di ranchi, seguendo il lavoro [7] e [8], 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 ranchi 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.

I grafici in figura 35 e 36 mostrano i risultati delle simulazioni, rispettivamente per quanto riguarda delay di accesso e energia consumata[1]. Si ricordi che, fissata la dimensione della DRAM, scopo di queste simulazioni è quello di determinare la coppia $\langle \#banks, \#dies \rangle$.

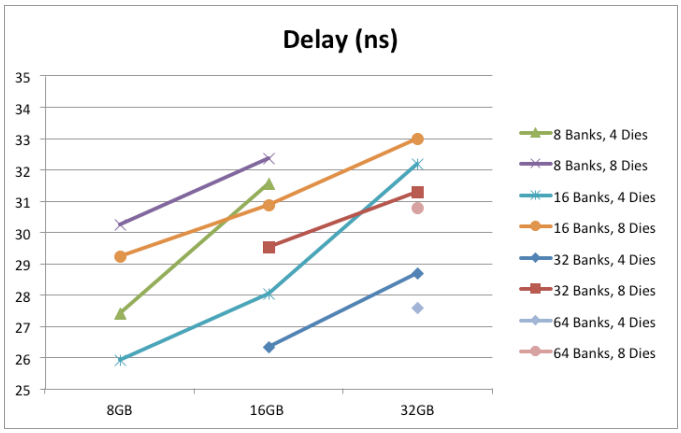

Figura 35: Latenze di accesso medie per ogni tipo di memoria considerate

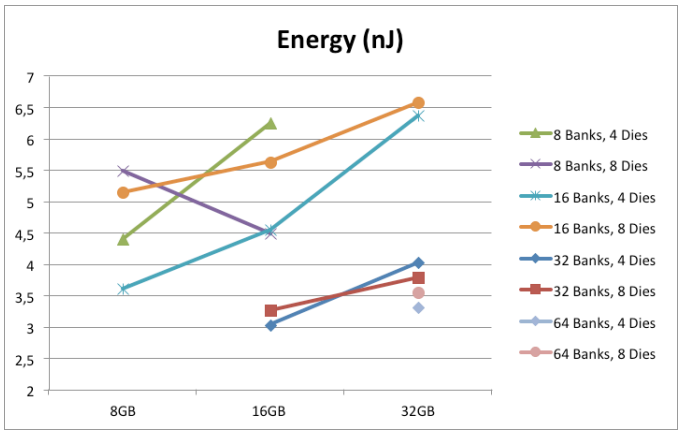

Figura 36: Energia consumata durante un accesso della durata media in DRAM

Una prima analisi della tabella 13 e dei grafici mostra come non tutte le combinazioni sono 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.

1 Per la valutazione viene considerata una operazioni di memoria media tra la più veloce e la più lenta, e tra la più dispendiosa e la meno in termini di consumo energetico

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.

12

PARAMETRI DI PROGETTAZIONE 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*, in [4], 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 in [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 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 calcogenico 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. I tempi di accesso reali simulati quindi risultano come segue :

```
// Time for a read (static)
#define t_READ 99.3243375
// Time for a write (static)
#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, è non 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 2D DRAM era utilizzata per la *writeback* 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 consentono di ridurre il fattore di scala da 4 a 1.3.

In tabella 14 vengono mostrati i risultati e dunque i parametri scelti per la alcune configurazioni di memoria PCM studiate in questo lavoro :

Nome Parametro	**Grandezza DRAM (GB)**	**Grandezza PCM (GB)**	**Fattore di scala**
Size	2	8	4
t_READ	24.8315ns	99.3243ns	4
t_WRITE	24.8315ns	794.5947ns	32
READ_ENERGY	0.862207nJ	1.1208691nJ	1.3
WRITE_ENERGY	0.862209nJ	1.1208717nJ	1.3

Tabella 14: Valori di progetto ottenuti per la PCM

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.

13

ARCHITETTURA PCM COME MAIN MEMORY E 3D-DRAM COME PART OF MAIN MEMORY

In questo capitolo verranno esaminate le simulazioni e i relativi risultati dell'architettura che prevede l'utilizzo di una memoria principale composta in parte da una supporto 3D DRAM ed in parte da un supporto PCM. Punti di forza di questa nuova architettura sono senz'altro l'incremento di performance ottenibili con una giusta scelta della politica di smistamemento, ma soprattutto la sua *scalabilità* grazie all'utilizzo del nuove e promettente dispositivo PCM.

Come già ampiamente trattato nel capitolo 5 questa architettura prevede due approcci ben distinti : nel primo si vorrebbe rendere il sistema operativo non conscio della divisione degli spazi di memoria e mentre nel secondo il sistema operativo assumerebbe un ruolo attivo nello smistamento delle operazioni di memoria tra i due dispositivi.

Ovviamente utilizzando il primo approccio ci dovrà occupare della progettazione di un controllore di memoria in grado di poter smistare opportunamente le operazioni di memoria, seguendo una politica definita. Sono state evidenziate 5 politiche principali da poter implementare in questo ambiene :

1. **Politica di smistamento a gruppi**
2. **Politica di smistamento a gruppi alternativa**
3. **Politica di smistamento forte**
4. **Politica di smistamento contiguo**
5. **Politica di smistamento basata sugli accessi alle pagine**

Tra queste abbiamo deciso di implementare la prima, ovvero la **politica di smistamento a gruppi** (si rimanda al paragrafo 5.1.1) per tre motivi :

1. non impone alcuna limitazione stringente sulla dimensione delle due memorie e sia più facilmente implementabile via hardware, a differenza dell'altra politica a gruppi (si rimanda al paragrafo 5.1.2) ;
2. si suppone che un'alternanza di pagine relative a due dispositivi sia da preferire rispetto ad una politca che prevede una divisione netta tra i due spazi di memoria (vedi politica 3 e 4) in quanto nell'eventualità che il *working set* dell'applicativo venga mappato in una sola delle due memorie (e probabilisticamente verrà mappato quasi interamente nella PCM) la valutazione dell'architettura nel suo insieme verrebbe notevolmente falsata;

3. questa politica risulta essere tra le prime 4 quella migliore per poter essere considerata una sorta di *baseline* per la valutazione di questa architettura in riferimento a tutte le possibile politiche *hardware-based* ma soprattutto in riferimento alla quinta politica che pensiamo possa portare al maggior incremento di performance per l'intero sistema.

Analizzando questa politica si noterà che il punto di forza sarà il rapporto tra le grandezze delle due memorie :

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

in quanto più sarà grande la parte di memoria gestita dalla 3D DRAM e più in linea teorica sarà performante il nostro sistema, dato che il dispositivo di memoria 3D risulta essere $\sim$ 8 volte più veloce per le operazioni di lettura e circa $\sim$ 30 volte per quanto riguarda le operazioni di scrittura. Come è facile intuire dalla figura x che presenta un esempio di implementazione della politica **politica di smistamento a gruppi** in cui il rapporto tra 3D DRAM e PCM risulta essere $\frac{1}{4}$ e quindi *gruppi* di 5 pagine fisiche di cui 1 relativa al dispositivo *veloce* e 4 al dispositivo più *lento*.

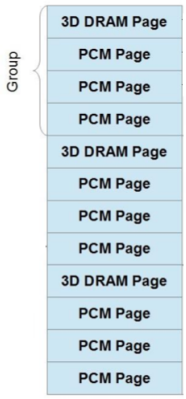

Figura 37: Algoritmo di smistamento a gruppi

E' facile dunque intuire che avere una porzione di memoria 3D DRAM più grande incrementerebbe la probabilità che l'indirizzo di memoria referenziato possa interessare la memoria più veloce.

Ovviamente senza un controllo diretto dell'assegnamento degli indirizzi fisici all'interno dei due dispositivi l'incremento di performance potrebbe non risultare eccellente in quanto fortemente dipendente dall'applicativo utilizzato per il *testing*, anche se come si vedrà si otterranno ottimi risultati.

13.1 MODIFICA DELL'AMBIENTE DI SIMULAZIONE

In questo capitolo verrà brevemente analizzata senza entrare nel dettaglio realizzato la modifica che è stata realizzata all'ambiente di simulazione per la specifica dell'architettura in trattazione.

Come già specificato nel capitolo 8 Simics permette la specifica di una architettura di memoria *a livelli*, partendo dalle cache di basso livello fino ad arrivare al dispositivo di memoria secondario. In figura 38 è mostrato lo schema a blocchi che descrive livello dopo livello come è stata definita l'architettura all'interno dell'ambiente di simulazione.

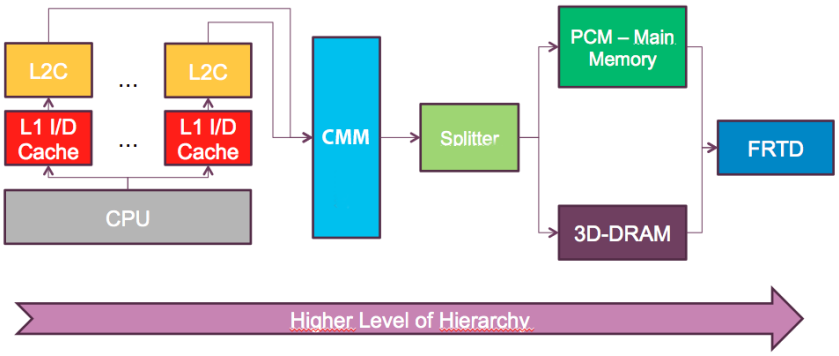

Figura 38: Flow-chart dell'architettura realizzata in Simics

Oltre alle *istruction* e *data* cache e al dispositivo *FRTD* di memoria secondaria, i blocchi fondamentali che costituiscono l'architettura sono :

- **CMM**
- **Splitter**
- **PCM - Main Memory**
- **3D-DRAM - Part of Main Memory**

Il primo blocco, il blocco **CMM**, è il blocco che implementa il paradigma *Cache Main Memory* con cui viene realizzata l'astrazione per il sistema operativo di una unica grande memoria unica, senza specificare la divisione dello spazio fisico in due parti distinte e implementa la gestione *cache-like* degli indirizzi di memoria effettuando la traduzione dell'indirizzo virtuale in indirizzo fisico, come già ampiamente specificato nel capitolo 3.

In questa nuova architettura però abbiamo già evidenziato la necessità di effettuare una ulteriore traduzione dell'indirizzo fisico in quello che abbiamo chiamato nel capitolo 5 *indirizzo fisico ibrido* di memoria, ovvero l'effettivo indirizzo fisico all'interno dei singoli dispositivi di memoria. Il modulo denominato **Splitter** è il componente incaricato di questa traduzione ed è forse il modulo più importante

in quanto all'interno di questo modulo è possibile specificare la politica di smistamento degli indirizzi. Nel particolare della politica di smistamento che abbiamo scelto di simulare, di seguito è riportato un estratto del modulo **Splitter** dove viene effettuato lo smistamento dell'indirizzo fisico e la sua ulteriore traduzione :

Algoritmo 13.1 Algoritmo estratto dal modulo **Splitter** che effettua lo smistamento e la traduzione dell'indirizzo fisico nell'indirizzo fisico ibrido di memoria

```
uint64_t DramPagesInABlock = splitter->config.DramMemorySize;
uint64_t PcmPagesInABlock = splitter->config.PcmMemorySize;
uint64_t TotalPagesInABlock = splitter->config.TotalMemorySize;
uint64_t address = 0;
uint64_t pageAddress = pa & 0xFFFFFFFFFFFF8000;
uint64_t blockIndex = pageAddress / TotalPagesInABlock;
uint64_t offset = pageAddress % TotalPagesInABlock;

if(splitter->DramTimingModel!=NULL && splitter->PcmTimingModel!=NULL)
{
      if (offset < DramPagesInABlock)
      {
      address = blockIndex * DramPagesInABlock + offset;
      address <<= 15;
      address |= (pa & 0x0000000000007FFF);
      cloneOp->s.physical_address = address;
      splitter->statistics.DramTransaction++;

      stall_time+=splitter->DramTimingInterface->operate(splitter->
          DramTimingModel, NULL, NULL, (generic_transaction_t*)
          cloneOp);
      }
      else
      {
      address = blockIndex * PcmPagesInABlock + (offset -
          DramPagesInABlock);
      address <<= 15;
      address |= (pa & 0x0000000000007FFF);
      cloneOp->s.physical_address = address;
      splitter->statistics.PcmTransaction++;

      stall_time += splitter->PcmTimingInterface->operate(splitter->
          PcmTimingModel, NULL, NULL, (generic_transaction_t*)cloneOp
          );
      }
```

Dopo aver effettuato la traduzione si dovrà calcolare il tempo in cui la CPU dovrà stallare ed attendere che l'operazione di memoria venga effettivamente portata a termine (si ricorda che nell'architettura CMM in caso di page faults non si verificano cambi di contesto ma l'unità di elaborazione attende il caricamento del dato in memoria). I componenti che realizzano questo ritardo sono i modulo **3D-DRAM** e **PCM**, che in Simics risultano essere ***timing models***, ovvero moduli addetti solamente al calcolo del tempo di *stallo* del processore in base al tipo di operazione di memoria e al dato di memoria stesso.

All'interno del blocco *if - else* si possono facilmente riconoscere le operazione per la realizzazione della politica di **smistamento a gruppi**, presentata nel paragrafo 5.1.1.

13.1.1 *Note realizzate del bus di memoria*

Prima di passare alla presentazione e alla analisi dei risultati vera e propria è necessario parlare brevemente del *bus* che collega il processore al dispositivo di memoria, che in ambienti reali introduce ritardi bassi ma non del tutto trascurabili ma che incidono fortemente sul consumo dell'intero sistema.

Abbiamo già studiato nel capitolo 1.2 la memoria 3D Stacked DRAM fa uso di un tipo di bus particolare per la comunicazione con la CPU e per la comunicazione *intra-banks*, i bus *Through Silicon Vias*. Le simulazioni effettuati con lo strumento CACTI presentate nel capitolo 11.4 forniscono parametri temporali e energitici per le operazioni di memoria che tengono conto del ritardo e del consumo introdotti dal bus *TSV*.

Tuttavia il bus che collega l'unità di elaborazione e il dispositivo di memoria PCM deve essere modellato a parte, ma in questa trattazione si è scelto di tralasciare questo aspetto in quanto anche nella configurazione utilizzata come **Baseline**, classica architettura di memoria principale 2D, non è stato preso in considerazione, e sarà oggetto di futuro sviluppo per tutte le architetture presentate in questo lavoro per ottenere risultati più accurati.

13.2 ANALISI DEI RISULTATI

Nei capitoli seguenti verranno analizzati i risultati ottenuti dal *testing* della architettura in trattazione con le suite di benchmark già esposte ampiamente nel capitolo 9.

Avendo a che fare con una unità di elaborazione composta da 4 core fisici, come presentato nel capitolo 10, abbiamo deciso di sottoporre il nostro sistema sotto *stress* utilizzando dei *mix* di benchmark tra tutti quelli a disposizione con workload crescenti sia in termini di *footprint* di memoria sia in termini di complessità computazionale.

Come è facile intuire i suddetti *mix* sono composti da 4 differenti benchmark, e nel nostro caso sono stati utilizzati tutti i *mix* già utilizzati in [5] per il *testing* dell'architettura CMM. La tebella 15 mostra i carichi di lavoro presi in considerazione per quanto riguarda la suite di benchmark SPEC CPU2006 :

Mix	Bench1	Bench2	Bench3	Bench4	Total (MB)
Small1	Gobmk	Hmmer	H264Ref	Gromacs	41.7
Small2	Gamess	Sphinx3	Tonto	Namd	27.6
Medium1	Sjeng	Libquantum	Leslie3d	Astar	191.6
Medium2	Omnetpp	Astar	Calculix	Gcc	139.9
Large1	Milc	Wrf	Zeusmp	Soplex	865.9
Large2	Zeusmp	Leslie3d	Gcc	CactusADM	718.3
VeryLarge1	GemsFDTD	Mcf	Bwaves	CactusADM	2262.2
VeryLarge2	Mcf	Zeusmp	Milc	Bwaves	1656.0

Tabella 15: Benchmark mixes e relativi workloads per la suite SPEC CPU2006

Questi *mixes* hanno le caratteristiche di avere un *footprint* non molto elevato ma un carico computazionale piuttosto *stressante*. E' stato quindi usato dapprima questo approccio per valutare le prestazione in termini di IPC e consumo di potenza del nostro sistema considerando un suo possibile utilizzo in server dedicati principalmente al calcolo scientifico.

In seconda analisi abbiamo presupposto l'utilizzo della architettura proposta in un ambito ben diverso, ovvero quello degli *High Performance Computers*, ovvero macchine che oltre ad esigere velocità consistenti in campi caratterizzati da complessi carichi computazionali, prevedono l'utilizzo di *footprint* di memoria molto elevati. La *scalabilità* della architettura in esame sembrerebbe adattarsi alla perfezione alla risoluzione di questo problema.

Per questo motivo sono stati utilizzati *cloud benchmarks* per un ulteriore *stressing* del sistema, in particolare la suite OLTP, già descritta nel capitolo 9.2. Anche in questo caso sono stati selezionati dei *mixes*, caratterizzati questa volta da un simile e piuttosto esteso *footprint* di memoria ma da campi di applicazione piuttosto differenti derivanti dalla natura dei benchmarks stessi : prenotazione di voli aerei, piattaforme di valutazione di prodotti, etc. .

In tabella 16 vengono presentati i *mixes* selezionati per la suite OLTP :

Mix	**Benchmark 1**	**Benchmark 2**	**Benchmark 2**	**Benchmark 2**	**Total (GB)**
AuctionMark	AuctionMark	Stream4.5	Stream4.5	Sjeng	$\sim$ 22.8
Seats	Seats	Stream4.5	Stream4.5	Sjeng	$\sim$ 23.4
Epinions	Epinions	Stream4.5	Stream4.5	Sjeng	$\sim$ 21.7
Tatp	Tatp	Stream4.5	Stream4.5	Sjeng	$\sim$ 24.3

Tabella 16: Benchmark mixes e relativi workloads per la suite OLTP

Come è facile notare all'interno di ogni *mix* considerato sono presenti due ulteriori *benchmarks*, **stream4.5** e **Sjeng**. La scelta di inserire questi ulteriori carichi è stata inspirata dal fatto che si è vo-

luto simulare anche un carico di *backend* all'interno della macchina simulata.

13.2.1 *Suite SPEC CPU2006 : IPC*

Per la valutazione prestazionale della nostra architettura con la *suite* SPEC CPU2006 abbiamo deciso di utilizzare solamente le 3 configurazioni che ci sembravano più adatte in termini di quantità totale di memoria (dati i ristretti *footprints* che caratterizzano questi *mixes*). Le 3 configurazioni sono quelle riportate in tabella 17 :

Memory Device	**No. 1**	**No. 2**	**No. 3**
3D-DRAM	2GB_4Banks 32KBpP_4Dies	4GB_8Banks 32KBpP_4Dies	8GB_16Banks 32KBpP_4Dies
PCM	8GB	8GB	8GB
Quantità totale di memoria (GB)	10GB	12 GB	16 GB

Tabella 17: Configurazioni hardware testate con la suite SPEC CPU2006

dove le 3D DRAM considerate sono quelle risultate migliori nello studio effettuato nel capitolo 11.4 e i valori delle PCM sono stati dedotti a partire da una memoria DRAM 2*D* planare d grandezza 2GB, secondo considerazioni già ampiamente trattate nel capitolo 12.

In figura 39 sono presentati gli andamenti dei valori di *Istruction Per Cycle* per ogni configurazione hardware sopra elencata al variare del *mix* di benchmark considerato.

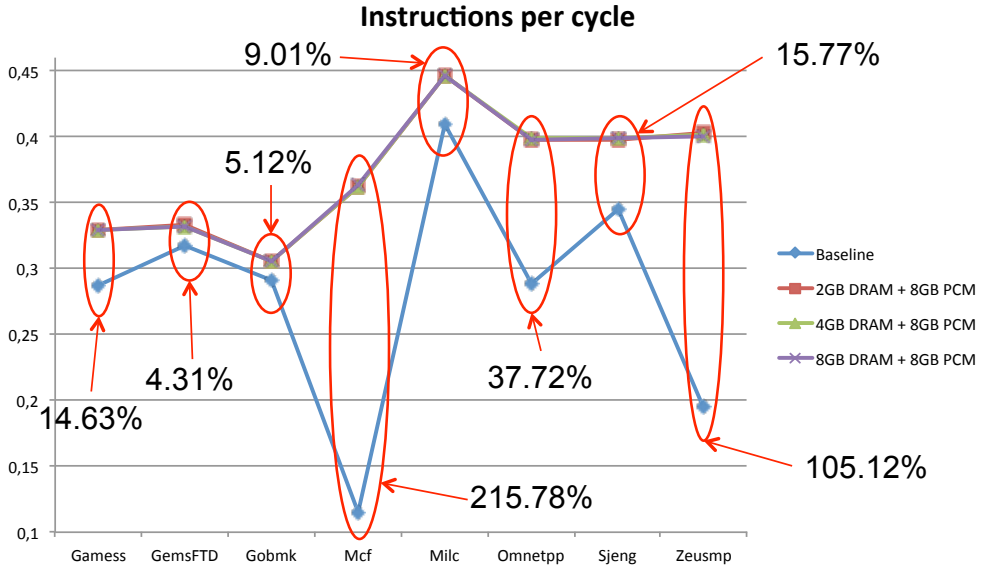

Figura 39: IPC per la suite SPEC CPU2006

I risultati confermano ampiamente quelli già ottenuti in [5] con la valutazione dell'architettura CMM, che avevano evidenziato un incremento di prestazioni medio pari al $\sim$ 33%. Tuttavia la configurazione trattata in questo lavoro sembra migliorare non di poco questo risultato, raggiungendo un incremento medio del $\sim$ 49%, soprattutto per quanto riguarda quei *mix* che prevedono *footprints* più elevati.

Nei capitoli conclusivi si cercherà di stimare quanto di questo forte incrementato possa essere attribuito alla architettura di memoria qua proposta e quanto sia da attribuire all'utilizzo della stessa nell'ambito di un sistema già molto rivoluzionato che adotta il sistema di memoria CMM.

13.2.2 *Suite SPEC CPU2006 : consumo di potenza*

In figura 40 vengono presentati, per ogni configurazione scelta nel capitolo 13.2.1 i consumi di potenza dell'architettura al variare del *mix* di benchmark.

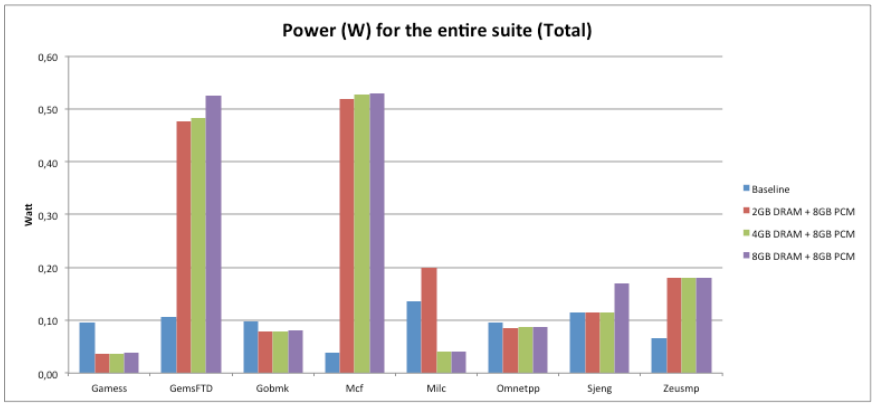

Figura 40: Potenza consumata per la suite SPEC CPU2006

Come è facile notare la diffrenza tra il consumo delle architetture studiate in questo lavoro e la *baseline* specificata nel capitolo 11.1 risulta limitata, se non per due mix che risultano molto meno performanti dal punto di vista energetico.

Questo tuttavia è un risultato che ci saremmo dovuti aspettare, in quanto la realizzazione dell'architettura CMM e in particolare del nostro sistema ibrido di memoria introduce tutta una serie di componenti aggiunti (SRAM, controllore ibrido di memoria) che non risultano essere ovviamente trascurabili nel computo generale. Inoltre si pone all'attenzione che mentre le energie relative ad operazioni di memoria tra 3D DRAM e 2D DRAM sono pressochè comparabili, l'energia consumata dal dispositivo di memoria PCM risulta essere circa $\sim$ 1.5 volte maggore (anche se caratteristica peculiare dei dispositivi PCM è la quasi totale assenza di *leakage*).

E' inoltre importante considerare che la potenza consumata da questa nuova architettura, anche nel caso dei due *mix* più 'dispendiosi'

risulta essere nettamente minore alla potenza consumata dall'unità di elaborazione presa in considerazione in questo lavoro, che varia dai $\sim 130W$ ai $\sim 200W$ in relazione ovviamente al carico di lavoro.

13.2.3 *Suite OLTP : IPC*

In questo capitolo vengono presentati i risultati ottenuti con la suite di benchmark OLTP per la valutazione della nostra architettura inserita in un ambiente *cloud* o *HPC*.

Per questi *stress-test* sono state prese in considerazione delle configurazioni hardware differenti da quelle studiate con la suite SPEC CPU2006 per i motivi già ampiamente discussi e riguardanti il limitato *footprint*.

Le configurazioni hardware prese in esame in questo capitolo sono riassunte in tabella 18 :

Memory Device	**No. 1**	**No. 2**	**No. 3**
3D-DRAM	2GB_4Banks 32KBpP_4Dies	4GB_8Banks 32KBpP_4Dies	8GB_16Banks 32KBpP_4Dies
PCM	16GB	16GB	16GB
Quantità totale di memoria (GB)	18GB	20GB	24GB

Memory Device	**No. 4**	**No. 5**	**No. 6**
3D-DRAM	2GB_4Banks 32KBpP_4Dies	4GB_8Banks 32KBpP_4Dies	8GB_16Banks 32KBpP_4Dies
PCM	32GB	32GB	32GB
Quantità totale di memoria (GB)	34GB	36GB	40GB

Memory Device	**No. 7**	**No. 8**
3D-DRAM	16GB_32Banks 32KBpP_4Dies	32GB_64Banks 32KBpP_4Dies
PCM	32GB	32GB
Quantità totale di memoria (GB)	48GB	64GB

Tabella 18: Configurazioni hardware testate con la suite OLTP

Come si vede facilmente abbiamo voluto spingere al massimo il rapporto tra le due memoria per riuscire a capire se, anche senza una oculata politica di smistamento delle pagine, si possono raggiungere ottimi risultati come quelli raggiunti con la suite SPEC CPU2006.

In figura 41 viene presentato il grafico relativo alla variazione di IPC tra i *mix* OLTP evidenziati nel capitolo 13.2.3 e il loro confronto con la baseline al variare della configurazione hardware.

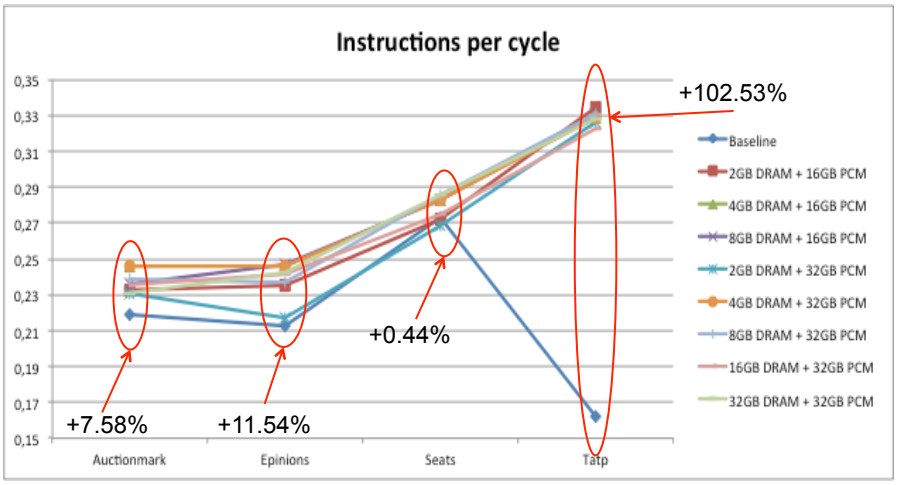

Figura 41: IPC per la suite OLTP

Come è facile notare a prima vista non si ottengono con questo tipo di suite gli stessi incrementi di performance che si sono registrati con la suite di benchmark precedenti. Questo è dovuto sicuramente al complesso comportamento del benchmark stesso, caratterizzato da un *footprint* decisamente esteso, e con molta probabilità alla politica di smistamento utilizzata.

Infatti come era prevedibile aspettarsi, le configurazioni in cui la quantità di memoria 3D DRAM risulta essere la metà o pari alla quantità di memoria PCM sono quelle che registrano il maggior incremento il termini di IPC. La media di incremento delle *IPC* relativamente alle varie configurazioni studiate e la baseline risulta essere di $\sim 30.52\%$, un valore molto promettente se anche paragonato ai risultati ottenuti in [10] in cui sono valutate le prestazioni dell'architettura CMM e dove era presente un solo dispositivo di memoria costituito dalla 3D DRAM.

13.2.4 *Suite OLTP : consumo di potenza*

In figura 42 vengono presentati, per ogni configurazione scelta nel capitolo 13.2.1 il consumo di potenza dell'architettura al variare del *mix* di benchmark.

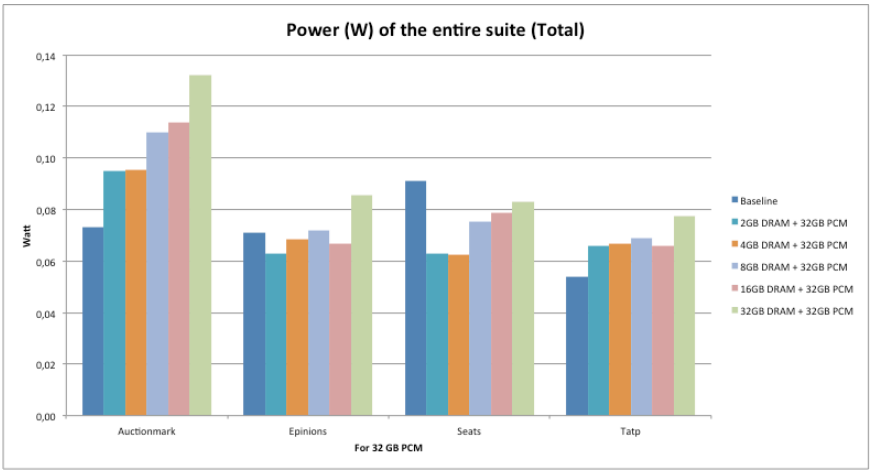

Figura 42: Potenza consumata per la suite OLTP

A differenza dei risultati ottenuti per la suite SPEC CPU2006 il consumo energetico della architettura studiata in relazione alla suite OLTP risulta molto simile (e in qualche caso persino migliore) alla configurazione scelta come *baseline*, a sola eccezione del *mix* AuctionMark. Questo è risultato molto convincente poichè il vero interesse per questa architettura risiede nel suo possibile utilizzo in sistemi *cloud* o *HPC*, ovvero sistemi perfettamente simulati da questi *mix* di benchmark OLTP.

13.3 SVILUPPI FUTURI

Come abbiamo visto questo tipo di architettura risulta essere molto promettente poichè estremamente *scalabile* e estremamente performante se paragonata ad una attuale architettura di memoria basata su una tecnologia ormai stagnante e caratterizzata da uno sviluppo lentissimo.

Ciononostante molteplici miglioramenti possono essere effettuati per incrementare ulteriormente questo aumento di performance. Primo fra tutti riteniamo possa essere l'implementazione della **politica di smistamento basata sugli accessi alle pagine**. Questo tipo di politica è già presente nelle architetture odierne e viene utilizzata nella scelta delle pagine *vittime* da dover trasferire dalla memoria principale verso la memoria secondaria e come è ben noto porta grandi benefici.

Si prevede che avere a disposizione infatti uno spazio di memoria che risiede nel dispositivo di archiviazione più veloce (la 3D DRAM) contenente le pagine più frequentemente accedute e prevedere un meccanismo di *swapping* delle pagine tra i due dispositivi incrementerebbe ulteriormente le prestazioni del sistema, sia in termini di IPC sia in termini di potenza consumata (si ricordi infatti che la 3D DRAM è caratterizzata da un consumo decisamente inferiore rispetto

ai dispositivi PCM).

Ulteriore conferma dell'estrema importanza della politica di smistamento all'interno di questa architettura viene fornito dal grafico 43 che riporta un esempio di come , considerando la suite SPEC CPU2006, pur avendo un rapporto 1 ad 1 tra le grandezze di 3D DRAM e PCM la maggior parte degli accessi effettuati si riferiscano al dispositivo PCM, andando così ad influire negativamente sulle prestazioni del sistema.

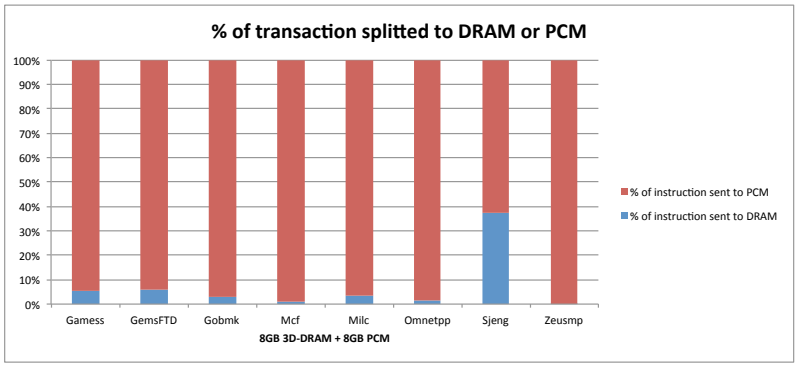

Figura 43: Percentuale di smistamento tra 3D DRAM e PCM degli accessi in memoria

Ulteriore concentrazione dovrebbe essere prossimamente posta nella **modellazione dei bus di memoria**, sia per la configurazione *baseline* sia per la nuova architettura studiata. Questo risulterebbe in una più accurata valutazione consumi energetici del sistema intero.

Ultimo ma non meno importante futuro sviluppo dovrà essere il **meccanismo di traduzione degli indirizzi virtuali in indirizzi fisici** al'interno dell'architettura CMM. Il fatto che adesso quest'ultima memoria *cache-like* sia indirizzata e *taggata* virtualmente causa sicuramente un scarsa ottimizzazione dell'assegnamento degli indirizzi di memoria e un impronta sicuramente molto rilevamente del fenomeno di *trashing* sui risultati delle simulazioni.

BIBLIOGRAFIA

[1] 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 Bryan Black, Murali Annavaram and Clair Webb. Die stacking (3d) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006. (Cited on page 3.)

[2] Ke Chen, Sheng Li, N. Muralimanohar, Jung-Ho Ahn, J.B. Brockman, and N.P. Jouppi. Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 33–38, 2012. doi: 10.1109/DATE.2012.6176428. (Cited on pages xi, 57, and 58.)

[3] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary dram architectures. In *Computer Architecture, 1999. Proceedings of the 26th International Symposium on*, pages 222–233, 1999. doi: 10.1109/ISCA.1999.765953. (Cited on page 23.)

[4] Xiangyu Dong, Cong Xu, Yuan Xie, and N.P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(7):994–1007, 2012. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2185930. (Cited on page 87.)

[5] Ademola Fawibe, Jared Sherman, Krishna Kavi, Mike Ignatowski, and David Mayhew. New memory organizations for 3d dram and pcms. In Andreas Herkersdorf, Kay Römer, and Uwe Brinkschulte, editors, *Architecture of Computing Systems – ARCS 2012*, volume 7179 of *Lecture Notes in Computer Science*, pages 200–211. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28292-8. doi: 10.1007/978-3-642-28293-5_17. URL http://dx.doi.org/10.1007/978-3-642-28293-5_17. (Cited on pages 13, 15, 17, 19, 20, 73, 78, 80, 93, and 96.)

[6] Gilbert Lecarpentier and Joeri De Vos. Die 2 die bonding, 2012. URL http://proxy.siteo.com.s3.amazonaws.com/www.set-sas.fr/file/dp2012-d2wbonding2012-03-08.pdf. (Cited on page 4.)

[7] G.H. Loh. 3d-stacked memory architectures for multi-core processors. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 453–464, 2008. doi: 10.1109/ISCA.2008.15. (Cited on pages xi, 3, 4, 6, 7, 58, and 83.)

[8] G.H. Loh. Computer architecture for die stacking. In *VLSI Technology, Systems, and Applications (VLSI-TSA), 2012 International Symposium on*, pages 1–2, 2012. doi: 10.1109/VLSI-TSA.2012.6210108. (Cited on page 83.)

[9] Mark S. Papamarcos and Janak H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. *SIGARCH Comput. Archit. News*, 12(3):348–354, January 1984. ISSN 0163-5964. doi: 10.1145/773453.808204. URL http://doi.acm.org/10.1145/773453.808204. (Cited on page 29.)

[10] Giandomenico Pisano. Progetto e valutazione delle prestazioni e dell'impatto energetico degli ultimi due livelli di una gerarchia di memorie ad alte prestazioni basata su tecnologie 3d-dram e pcm. Master's thesis, University of Pisa, 2013. (Cited on pages 19 and 98.)

[11] Quereshi. *Phase Change Memory ñ from Devices to Systems*. Morgan & Claypool Publishers, 2011. (Cited on pages xi, 10, and 11.)

[12] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. *SIGARCH Comput. Archit. News*, 37(3):24–33, June 2009. ISSN 0163-5964. doi: 10.1145/1555815.1555760. URL http://doi.acm.org/10.1145/1555815.1555760. (Cited on pages xi, 25, 32, 34, and 87.)

[13] S. Raoux, G.W. Burr, M.J. Breitwisch, C.T. Rettner, Y.C. Chen, R.M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. L Lung, and C.H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465–479, 2008. ISSN 0018-8646. doi: 10.1147/rd.524.0465. (Cited on page 9.)

[14] Giuseppe Regina. Progettazione e analisi di una memoria principale ad alte prestazioni basata su pcm e di una cache realizzata con 3d-dram. Master's thesis, University of Pisa, 2013. (Cited on page 19.)

[15] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *University of Virginia*, 1:1–3, 1994. (Cited on page 1.)