Efficient Mapping of Multi-threaded Workloads on to Chip Multiprocessors

Thesis submitted in partial fulfilment of the requirements for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Rakesh Pandey

Under the supervision of

Dr. Aryabartta Sahu

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

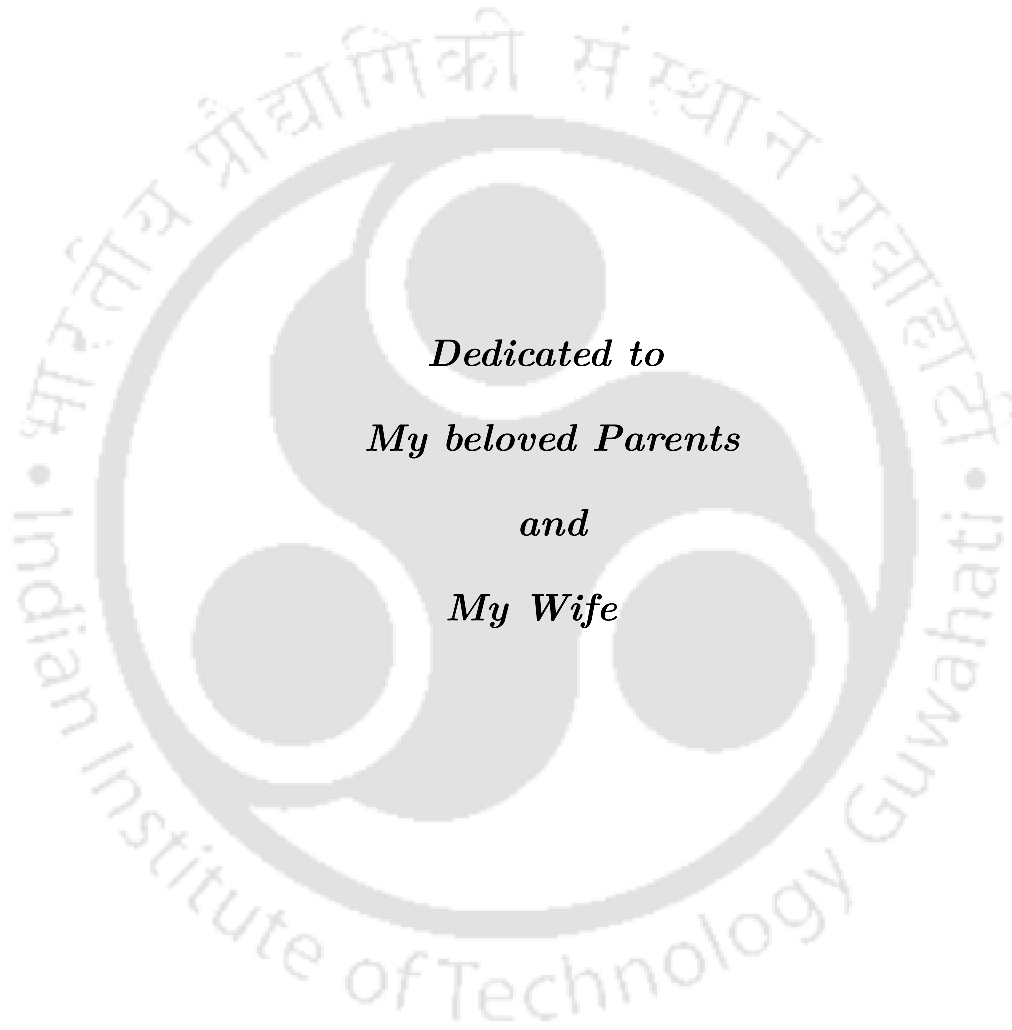
Guwahati - 781039 Assam India

September, 2019

TH-2297_136201001

Copyright © Rakesh Pandey 2019. All Rights Reserved.

TH-2297_136201001



Dedicated to

My beloved Parents

and

My Wife

TH-2297_136201001

TH-2297_136201001

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor *Dr. Aryabartta Sahu* for his unwavering support, inexhaustible patience and positive guidance during my doctoral research. I am thankful for his ethical beliefs and philosophy which made me mature as a scientific researcher.

I would also like to thank the rest of my thesis doctoral committee members - *Prof. Jatindra Kumar Deka*, *Prof. Hemangee K. Kapoor* and *Dr. Santosh Biswas* for their insightful comments and suggestions which made me improve the quality and clarity of my thesis work.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IIT Guwahati - Prof. Diganta Goswami and Prof. S. V. Rao for allowing me to use the facilities and the available resources including the travel support for the conferences. I am deeply thankful to - Mr. Monojit Bhattacharjee and Ms. Gauri Khuttiya Deori for efficiently handling the administrative work. I am obliged to all the faculty members, the staff and security personnel for their constant help and support.

I would also like to mention my seniors - Dr. Shashi Shekhar Jha, Dr. Shirshendu Das, Dr. Satish, Dr. Sibaji, Dr. Manojit Ghose, Dr. Mayank Agarwal, Dr. B. Bhaumik, R. Shenko, and Awnish for being my mentor-cum-friends and creating indelible moments at IIT Guwahati. I am fortunate to have good friends - Vasudevan, Achyut, Tushar, Sonia, Vidyapu, Pradeep Sharma and Ashish Rajak with whom I have shared some ineffaceable moments of my life at IIT Guwahati. I am thankful to all my colleagues and friends during my journey as a Ph.D. scholar.

Finally yet importantly, I would like to thank Almighty God and my family - Mummy, Papa, my wife Ajita and my little son Avyang for their boundless love, support, caring, warmth and encouragement all these years. I am truly indebted to them.

September 24, 2019 Rakesh Pandey

TH-2297_136201001

TH-2297_136201001

Declaration

I certify that

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor are not in a position to check for any possible instance of plagiarism within this submitted work.

September 24, 2019 Rakesh Pandey

TH-2297_136201001

TH-2297_136201001

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Dr. Aryabartta Sahu
Associate Professor
Email : asahu@iitg.ac.in
Phone : +91-361-2582370

Certificate

This is to certify that this thesis entitled "**Efficient Mapping of Multi-threaded Workloads on to Chip Multiprocessors**" submitted by **Rakesh Pandey**, in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy, to the Indian Institute of Technology Guwahati, Assam, India, is a record of the bonafide research work carried out by him under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam, India. To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date: September 24, 2019
Place: Guwahati

Dr. Aryabartta Sahu
(Thesis Supervisor)

TH-2297_136201001

TH-2297_136201001

ABSTRACT

Technology advancement in the area of IC design allows billions of transistors to be on a single chip, which allows the developments of the modern days' chip multiprocessors with larger core counts (in range of 100 cores or more). The increased core count in a chip multiprocessor urges the necessity of the high bandwidth memory, and high-speed on-chip interconnects. To fulfill the needs of the chip multiprocessors, many future generations on-chip interconnects as well as memory designs have been proposed.

Once the chip multiprocessor is fabricated, we need to use it efficiently. Therefore, to utilize the capabilities of the chip multiprocessors and get the best possible performance, application mapping emerged as a prominent area in the domain of chip multiprocessor research. In the past, application mapping on to the chip multiprocessors has mainly considered the task to core mapping, and not the data to memories, as on-chip memory was smaller (in megabytes). However, for the current as well as future generation chip multiprocessors where modern network-on-chip organizations and 3D-stacked memories have been proposed, investigating the impact of the application mapping and associated data placement becomes crucial. Therefore, this thesis proposes the techniques to efficiently map the applications on to the chip-multiprocessors considering the different current as well as future architectural design variations of the chip multiprocessor systems to improve the system performances.

This thesis takes a bottom-up approach and presents its first contribution to laid a foundation of the application mapping, and the conclusions arrived from this are used in the next contributions of this thesis. In the first contribution, a static profile based multi-threaded application mapping has been performed for the 3D-stacked DRAM memory based chip multiprocessor, where we consider the task to core mapping and virtual page to memory mapping techniques to improve the performance. This contribution considers the on-chip communication cost as the performance metric while evaluating the proposed techniques. Experiments show that the overall on-chip communication cost reduction due to the page mapping is significantly higher as compared to the reduction due to the task mapping. Moreover, virtual page to memory and task to core mapping reduces overall on-chip communication cost up to 86% (average 56%) and 26% (average 12%) respectively.

The conclusion of the first contribution along with the facts (a) task migration is a costlier operation and (b) most of the application shows phase-wise behavior at the run-time, motivated us to propose a self-adaptive run-time page mapping technique

TH-2297_136201001

as the second contribution of this thesis. Further, in the second contribution, we have performed a comparison between the proposed method along with an auxiliary SRAM buffer and a recent state-of-art work. Our experimental result shows that the proposed method can be *an alternative way to use the 3D-stacked DRAM memory for current as well as future chip multiprocessors systems*. The proposed self-adaptive run-time page mapping alone shows the communication cost reduction up to a maximum of 80% and an average of about 40% as compared to the base case method. Further, our self-adaptive run-time page mapping together with the SRAM mapping buffer outperforms the base-case by an average of 48% in terms of overall execution time. Also, most importantly, the adaptive run-time mapping with the SRAM mapping buffer shows a performance improvement by an average of 40% (in terms of overall execution time) when compared with state-of-art work where 3D-stacked DRAM used as a coherent cache with temporal SRAM buffer.

In the third contribution, considering the 3D-stacked DRAM-PCM hybrid memory as a viable alternative of the 3D-stacked DRAM memory, we have proposed an access-aware self-adaptive run-time page mapping for the 3D-stacked hybrid DRAM-PCM memory based target chip multiprocessor system. Our technique minimizes the DRAM refresh related power consumption by performing a simple DRAM access-aware page placement between DRAM and PCM of the hybrid memory slice. Further, it uses the DRAM row access information and performs an access-aware self-adaptive page mapping for the optimized page placement between the different hybrid memory modules of the 3D-stacked hybrid memory. Our proposed approach performs similar or better in terms of the execution time and reduces the energy consumption due to the DRAM refresh by an average of 51% as compared to the base case.

In the fourth and final contribution, we perform the trade-off analysis between the performance and cache size of the chip multiprocessor system while utilizing the benefits of the high-end optical interconnects, 3D-stacked DRAM memory and a self-adaptive run-time page mapping (as proposed in the second contribution). In this contribution, we found that for a fixed chip multiprocessor die size, reducing the cache size per core increases the on-chip communication cost and decreases the system instruction per cycle. However, the chip multiprocessor performance degradation due to the reduction in cache size per core can be nullified with the use of an efficient hybrid interconnection network, 3D-stacked DRAM memory, and self-adaptive run-time page mapping.

TH-2297_136201001

Contents

List of Figures xviii

List of Algorithms xix

List of Tables xxi

List of Symbols xxiii

List of Abbreviations xxv

1 Introduction 1

1.1 Chip Multiprocessor . . . 2

1.1.1 3D-stacked On-chip DRAM Memory . . . 3

1.1.2 3D-stacked On-chip Hybrid DRAM-NVRAM Memory . . . 3

1.1.3 Network-On-Chip . . . 4

1.1.4 Hybrid Network-On-Chip . . . 5

1.2 Multi-threaded Application and Run-time Phase-wise Behavior . . . 6

1.3 Motivation . . . 9

1.4 Objectives . . . 11

1.4.1 Static Profile Based Mapping . . . 11

1.4.2 Run-time Dynamic Mapping . . . 12

1.4.3 Run-time Mapping Considering Hybrid Memory . . . 13

1.4.4 Performance Analysis of CMP having 3D-stacked DRAM and Hybrid NOC . . . 13

1.5 Contributions . . . 14

1.5.1 Static Profile Based Mapping . . . 14

1.5.2 Self-adaptive Run-time Page Mapping . . . 15

1.5.3 Run-time Page Mapping Considering Hybrid Memory . . . 16

1.5.4 Performance Analysis of CMP having 3D-stacked DRAM and Hybrid NOC 17
1.6 Thesis Organization 18

2 Related Works 19
2.1 Application Mapping 19
2.2 3D-stacked DRAM Memory 21
2.3 3D-stacked Hybrid Memory 23
2.4 Network-On-Chip 24
2.5 Hybrid Network-On-Chip 25
2.6 Area and Performance Trade-off Implication Using Hybrid NOC and 3D-stacked Memory 26

3 System Model and Application Model 29
3.1 System Model and its Variations 29
3.1.1 DRAM Memory at the 3D-stacked Memory Layer 33
3.1.2 DRAM and SRAM Buffer at the 3D-stacked Memory Layer . 34
3.1.3 DRAM and PCM memory at the 3D-stacked Memory Layer . 36
3.1.4 DRAM Memory at the 3D-stacked Memory Layer along with an Optical Layer 37
3.2 Application Model 39

4 Static Profile Based Mapping 41
4.1 Problem Formulation 42
4.2 Static Profile Based Mapping 43
4.2.1 Thread to Core Mapping 44
4.2.2 Mapping of Virtual Pages to DRAM Memory Slices 45
4.2.3 Thread Mapping Followed by Virtual Page Mapping 47
4.2.4 Combined Thread Mapping and Virtual Page Mapping . . . 47
4.3 Experimental Setup 49
4.4 Result and Overhead Analysis 51
4.4.1 Result Analysis 51
4.4.2 Overhead Analysis 55
4.5 Summary 56

TH-2297_136201001

5 Self-adaptive Run-time Page Mapping **57**
5.1 Problem Formulation 58
5.2 Self-adaptive Run-time Page Mapping 61
5.2.1 Page Access and Run-Time Profiling 62
5.2.2 Page Mapping, Migration and TLB Update 63
5.2.3 Experimental Setup 67
5.2.4 Results 68
5.3 Comparison with Coherent DRAM Cache 70
5.3.1 Results 72
5.4 Performance and Area Overhead 77
5.5 Summary 79

6 Run-time Page Mapping Considering Hybrid Memory **81**
6.1 Access-Aware Self-adaptive Page Mapping on to Hybrid Memory Slices 84
6.1.1 Page Access and Run-Time Profiling 85
6.1.2 Page Mapping Decision Making 87
6.2 Access-Aware Page Placement Between DRAM and PCM of the Hybrid Memory Slice 90
6.3 Experimental Result and Analysis 92
6.4 Performance and Area Overhead Analysis 95
6.4.1 Performance Analysis 95
6.4.2 Area Overhead Analysis 97
6.5 Summary 98

7 Performance Analysis of CMP having 3D-stacked DRAM and Hybrid NOC **99**
7.1 Target System Architecture 100
7.1.1 Routing of Packets 101
7.2 Problem Formulation 103
7.3 Self-adaptive Application Mapping 108
7.4 Experimental Environment 109
7.5 Results 109
7.6 Performance and Area Overheads 115
7.6.1 Performance Overheads 115
7.6.2 Area Overhead 116

TH-2297_136201001

7.7 Summary 117

8 Conclusions and Future Perspectives 119

8.1 Summary of Thesis 120

8.2 Future Research Avenues 122

Publications 125

Vitae 127

TH-2297_136201001

List of Figures

1.1 Example of 2D-mesh based industry level CMPs 5

3.1 Example: 3D representation of the system model 30
3.2 Detailed system model representation showing four non-overlapping areas. 31
3.3 Example: Detailed system model representation for CMP with DRAM memory 34
3.4 Example: Detailed system model representation for CMP with DRAM memory and SRAM buffer 35
3.5 Example: Detailed system model representation for CMP with DRAM-PCM based hybrid memory 36
3.6 Example: Detailed system model representation for CMP with DRAM memory and optical interconnect 37
3.7 Example of application model 39

4.1 Percentage of active virtual pages mapped to different memory slices: before and after page mapping 47
4.2 Normalized communication cost on 4×4 CMP system 52
4.3 Normalized communication cost on 6×6 CMP system 53
4.4 Normalized communication cost on 8×8 CMP system 53
4.5 Normalized communication cost for multi-application workload, where f, c, h, l, g and m are fft, cholesky, heat, lu, magic and matmul benchmark respectively from Cilk 54
4.6 Normalized overall execution time on 8×8 CMP system 56

5.1 Memory controller (naive MC with new mapping hardware) 63
5.2 Shared centralized TLB and distributed private TLB organization of a 4×4 system 65
5.3 Dynamic run-time page mapping considering 4×4 CMP 69

5.4 Overview of 16KB $Mbuff$ 72
5.5 Overall normalized execution time for different benchmarks, average is reported as "AVG" 74
5.6 Mapping-buffer hit rate for its different size 75
5.7 Overall normalized execution time considering different L_{DRAM} and L_{H2H} values, average is reported as "AVG" 76

6.1 DRAM refresh power consumption for DRAM devices [79]. 82
6.2 Hybrid memory module architecture 83
6.3 Relation between an epoch and row monitoring period 84
6.4 Memory controller (naive MC with new mapping hardware) 86
6.5 DRAM refresh controller architect inside the modified MC 91
6.6 Normalized execution time 96
6.7 Normalized energy consumption considering the DRAM refresh energy and PCM access energy 96

7.1 CMP system architecture. 101
7.2 Example of $CMHIG$ graph 103

TH-2297_136201001

List of Algorithms

1	:Thread to Core Mapping Using Simulated Annealing	45
2	:Virtual Pages to DRAM Slice Mapping	46
3	:Combined Virtual Page and Thread Mapping	50
4	:Page Access and Run-time Profiling	63
5	:Page Mapping Decision Making	64
6	:Page Access and Run-time Profiling for Hybrid Memory	87
7	:Page Mapping Decision Making for Hybrid Memory	88
8	:Working of the DRAM Refresh Controller Inside Each Modified Memory Controller	93

TH-2297_136201001

TH-2297_136201001

List of Tables

4.1 Initial and optimized thread to core mapping 45
4.2 Default CMP system configuration parameters 52

5.1 System configuration parameters 68
5.2 Communication cost (CCost) overhead of different TLB organization and policies for 4×4 2D mesh, due to respective operation 69
5.3 Example: Total number of L2 cache misses and page migration . . . 78
5.4 Area overhead summary. 79

6.1 System configuration parameters 94

7.1 A segment of the VIA table stored at router of the NOC-tile vertex c_0, where: $\mathbf{IN_{src}}$ and $\mathbf{IN_{dst}}$ are intermediate nodes adjacent to source and destination nodes respectively. 103
7.2 Number of cores that can fit with different cache and core size for the CMP die area of $240mm^2$, using (7.1). 105
7.3 System configuration parameters. 110
7.4 Summary of the benchmarks characteristics 110
7.5 Example: Total number of L2 cache misses and page migrations. . . . 116

TH-2297_136201001

List of Symbols

Symbols	Description
N	Number of processor cores
M	Number of memory slices
$\mathbf{M}$	Set of memory slices
m_i	i^{th} memory slice, $\forall\ m_i \in \mathbf{M}$
CST	Core to slice array
$SNum$	Memory slice number indexed by CST array
$CMIG$	Core memory interconnection graph
$\mathbf{E_{Ecc}}$	Electrical edge set
$\mathbf{C_{cc}}$	Set of NOC-tile vertices (or core vertices or cores)
c_i	i^{th} NOC-tile vertex (or core vertex or core)
$e_{ecc}(c_i, c_j)$	Electrical edge between cores c_i and c_j
$Mbuff$	SRAM based buffer (termed as mapping buffer)
$\mathbf{ONI}$	Set of optical network interfaces
oni_i	i^{th} optical network interface, $\forall\ oni_i \in \mathbf{ONI}$
$CMHIG$	Core memory hybrid interconnection graph
$e_{occ}(c_i, c_j)$	Optical edge between some cores c_i and c_j
AG	Application graph
$AGVP$	Application graph with virtual paging
$\mathbf{T}$	Set of threads (or thread nodes)
t_i	i^{th} thread (or thread node), $\forall\ t_i \in \mathbf{T}$
$\mathbf{D}$	Set of data nodes
d_i	i^{th} data node, $\forall\ d_i \in \mathbf{D}$
$\mathbf{E_{tc}}$	Set of thread communication edges
$e_{tc}(t_i, t_j)$	A thread communication edge, $\forall\ e_{tc}(t_i, t_j) \in \mathbf{E_{tc}}$
$\mathbf{E_{da}}$	Set of data access edges
$e_{da}(t_i, d_j)$	A data access edge, $\forall\ e_{da}(t_i, d_j) \in \mathbf{E_{da}}$

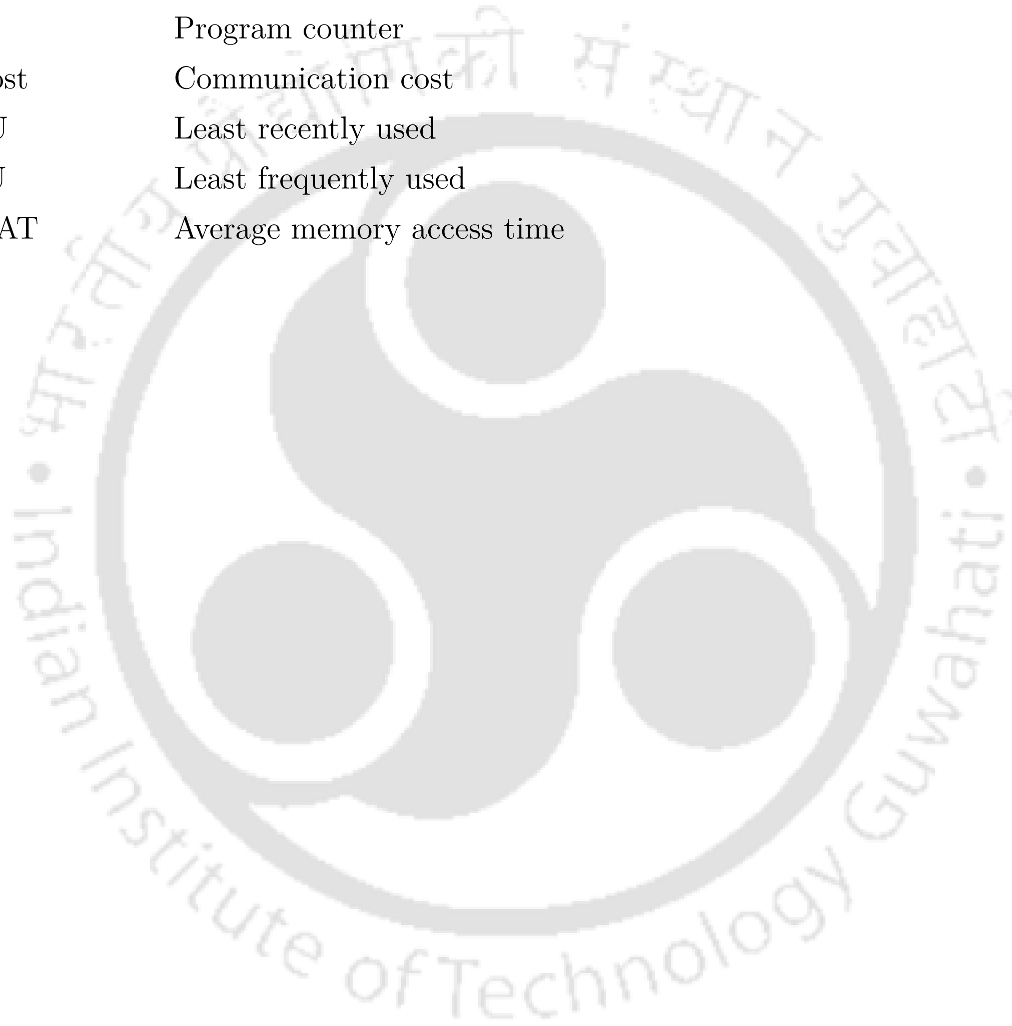
PP	Set of all physical pages (or frames)
pp_j	j^{th} physical page (or frame), $\forall\ pp_j \in \mathbf{PP}$
VP	Set of all virtual pages
vp_j	j^{th} virtual page, $\forall\ vp_j \in \mathbf{VP}$
$\mathbf{E_{vpa}}$	Set of virtual page access edges
$e_{vpa}(t_i, vp_j)$	A virtual page access edge, $\forall\ e_{vpa}(t_i, vp_j) \in \mathbf{E_{vpa}}$
C_{comm}	Overall thread communication cost
C_{mac}	Overall memory access cost
$distCC$	Manhattan distance between the core vertices
$X(t_i)$	Core vertices having mapped threads t_i
$\omega(e_{tc}(t_i, t_j))$	Amount of data communicated between thread t_i and t_j
$\omega(e_{vpa}(t_i, t_j))$	Number of page access by thread t_i to page vp_j
$Y(vp_j)$	Nearest core vertex to the memory slice having page vp_j
$T_{miss}(t_i, vp_j)$	Latency associated to thread t_i for the page vp_j at time t
$Y_t(vp_j)$	Nearest core vertex to the memory slice having page vp_j at time t
L_{MB}	$Mbuff$ access latency to access a cache block
L_{DRAM}	DRAM access latency to access a cache block of a page
MR_{MB}	$Mbuff$ miss rate
L_{H2H}	Hop to hop traversal latency
epo	A fixed time epoch (or phase) duration of a multi-phase application
$T_{epo,i}$	Latency to thread t_i for the page vp_j in a time epoch epo
$T_{Overall,i}$	Latency to thread t_i for all the pages during the total execution
$T_{L2M,total}$	Total miss latency of the application execution
Mig	Migrated page list
Cor_{Thr}	Parameter value
Loc	Target DRAM slice location list
C_{cntr}	Group of current epoch profile-counters
P_{cntr}	Group of previous epoch profile-counters
Cor	Correlation between profiling counters C_{cntr} and P_{cntr}

List of Abbreviations

Terms	Abbreviations
CMP	Chip multiprocessor
NOC	Network-on-chip
TSV	Through-silicon-vias
IPC	Instruction per cycle
IC	Integrated circuit
MB	Megabyte
GB	Gegabyte
KB	Kilobyte
MC	Memory controller
QAP	Quadratic assignment problem
SA	Simulated annealing
DRAM	Dynamic random-access memory
PCM	Phase-change memory
NVRAM	Non-volatile random-access memory
NVM	Non-volatile memories
SRAM	Static random-access memory
LPRAM	Logic process random-access memory
MCDRAM	Multi-channel dynamic random-access memory
3D	Three-dimensional
2D	Two-dimensional
LLC	Last level cache
I/O	Input-output
TLB	Translation lookaside buffer
MHz	Megahertz
GHz	Gegahertz
ONI	Optical network interface

TH-2297_136201001

SR	Specific-router
GR	General-router
OS	Operating system
ASLR	Address space layout randomization
VMT	Virtual page to memory slice table
TCT	Thread-to-core-mapping table
ACO	Ant colony optimization
PC	Program counter
CCost	Communication cost
LRU	Least recently used
LFU	Least frequently used
AMAT	Average memory access time



TH-2297_136201001

1

Introduction

In 1971, Intel has launched the first general-purpose programmable processor 4004 in the market with an advertisement in the November 15, 1971, issue of Electronic News: “Announcing A New Era In Integrated Electronics”. In 1971, the Intel 4004 processor had 2,300 transistors; and by 2010, an Intel Core processor with a 32 nm processing die held 560 million transistors within it [3].

Advancement in the integrated circuit (IC) fabrication technology have increased the transistor count significantly by fixing the processor size almost constant. Thermal losses occur when we put several billions of transistors together on a small area and switching them on and off again at several billion times per second speed. The faster we switch the transistors, the more heat generates and hence the thermal losses. Another drawback of the increased clock speed to switch the transistor on/off is that processor needs more voltage to work, and there exists a cubic dependency between the clock speed and the power consumption. Therefore, due to these significant problems, the processor clock speed became almost stagnant after reaching a specific value.

Techniques such as parallel processing, data-level parallelism, and instruction-level parallelism have been evolved and proven to be very effective to improve the performance and address the issue of power consumption [45]. Multi-core processor or chip multiprocessor (CMP) utilizes the parallelism to improve the performance and addresses the major challenges such as power efficiency and heat dissipation. These advantages are due to the fact that CMPs are constructed using simpler processors having lower voltage and frequency [84]. Moreover, the advancement across many

TH-2297_136201001

application domains, including network, embedded, and graphics domain requires a higher number of cores in the system to improve the performance. Olukotun *et al.* in [84], have also mentioned the reason to build the CMP system as,

> "The motivation for building a single-chip multiprocessor comes from two sources; there are a technology push and an application pull."

Following this notion, in this thesis, we consider some modern paradigms under the umbrella of the CMP system. Moreover, for pedagogical reasons, we briefly discuss the current and future generation CMP system; and introduce multi-threaded applications and their general behavior in the following section 1.1 and 1.2 respectively.

1.1 Chip Multiprocessor

During the past few years, there is enormous growth in the complexity of the multicore processor or CMP system implementation [115, 83, 54, 12]. Several industry vendors have started producing CMP at commercial levels, for example, Intel's core processors, Intel's Xeon Phi Knight Landing [107], Tilera's TILE64 [12], Qualcomm's mobile processors [4], AMD's Ryzen processor [1]. Traditionally, the design space exploration for CMP has focused on the computational aspects of the processing cores. However, as the number of cores on the CMP is increasing to fulfill the applications performance demand, consideration of the communication architecture and on-chip memory beyond the processor computational aspect became important parameters.

Initially when the number of cores on a CMP was small (< 10), the traditional design with small SRAM based cache (in MBs) and bus-based (or point-to-point) network has performed well. However, in the many-core era, as the number of cores residing on a CMP system is increasing rapidly, the performance of the system is getting limited by many architectural constraints and resource management techniques. In [100], Rogers *et al.* have found that off-chip memory bandwidth can severely restrict the core count of the CMP and thereby reduces the performance. Also, technology scaling in the CMP can cause unpredictable delays and high power consumption due to the in-efficient on-chip interconnection [50]. Researches have identified that network interconnect latency overhead is about 60% to 75% of the overall miss latency [102]; along with the power consumption of about 40% [38] of

TH-2297_136201001

the overall power consumption. Therefore, the CMP system is also urging the necessity of the low-latency, low-power, and high bandwidth on-chip interconnects or network-on-chip (NOC) [33].

1.1.1 3D-stacked On-chip DRAM Memory

In large CMPs (where the number of cores inside the chip is >10), the on-chip memory bandwidth requirement is very high. In [100], their study shows that on-chip memory (cache) size needs to grow much faster than the number of cores to compensate for the limited off-chip bandwidth. Feasible and cost-effective 3D fabrication technology prompted the researchers to integrate the future CMP systems with the architectures composed of the heterogeneous technologies. In the 3D chip design, layers fabricated using different techniques can be stacked over each other. It allows us to stack DRAM memory, non-volatile memories (NVM) (including phase-change memory, magnetic random access memories, etc.), and optical layer on top of the chip in 3D-stacked manner [20, 112, 126].

In the modern CMP system, to address the system bandwidth and performance demand, 3D-stacked DRAM memory is considered as a reassuring candidate [69, 55, 30]. 3D-stacked DRAM memory having through-silicon-vias (TSVs) provides higher bandwidth on-chip memory as well as a wide range of design flexibility to the CMP systems. Logic process RAM (LPRAM) design uses the same process, which is used to fabricate the processor or logic die. Stacking memory (or cache) on top of a multi-core die made feasible using Logic process RAM design. Examples of such systems are Blue Gene/L supercomputer and graphics synthesizer unit of Sony PlayStation2 which uses on-chip embedded DRAM multiprocessor. As reported in [67, 40, 69], placing DRAM main memory on top of the processor using 3D fabrication technologies (that is 3D-stacked memories) shows an impressive performance benefit up to 92%. Recently, Intel has developed Multi-Channel DRAM (MCDRAM), a 3D-stacked DRAM memory which is having $\approx 4\times$ more bandwidth as compared to the DDR4 [2].

1.1.2 3D-stacked On-chip Hybrid DRAM-NVRAM Memory

Although the 3D-stacked DRAM memory has various advantages, its power consumption is rapidly growing with the capacity increase. DRAM memory consumes 20% to

TH-2297_136201001

40% of the overall system power [66, 111]. Liu *et al.* in [68] have shown that power consumption due to the DRAM refreshes is projected to reach up to the 50% of the overall DRAM power consumption. Many previous studies have proposed techniques to reduce power consumption incurred due to the DRAM refresh operation [44, 6, 81].

To tackle the DRAM refresh power overhead associated with the larger size DRAM, phase change memory (PCM) has been explored as an alternative to the DRAM [129, 99, 97, 116]. Phase change memory (PCM) has better scalability (or higher density) and lower leakage energy as compared to the DRAM memory. Despite many advantages, PCM has some drawbacks as compared to the DRAM memory, for example, higher read/write latency and lower write endurance.

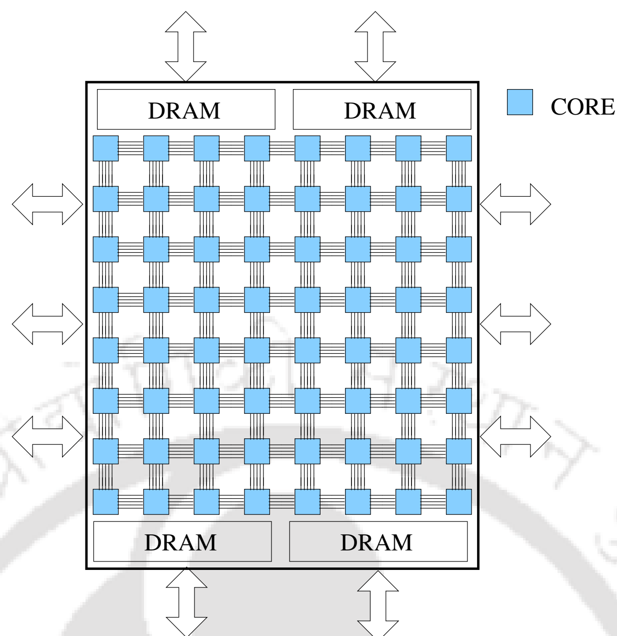
Therefore, hybrid memory designed using DRAM and PCM has been proposed as a potential solution, to take the benefits such as high capacity, lower power and better performance from both the technologies [64, 124, 126, 96]. Researchers in [64, 124] have studied the hybrid memory where DRAM works as a cache for the PCM. Whereas, in [126, 96] researchers have used DRAM and PCM components of the hybrid memory as two separate memories, and they can be accessed in parallel while holding a portion of the data.

1.1.3 Network-On-Chip

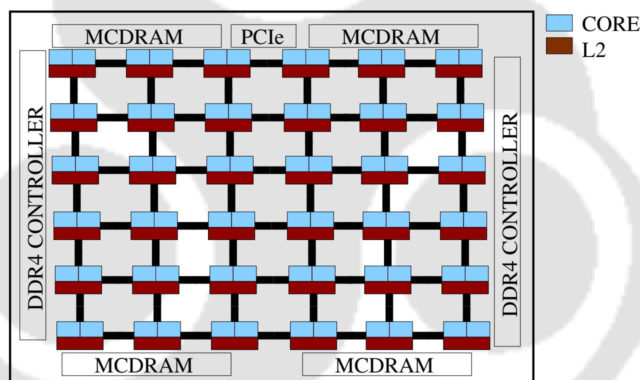
To fulfill the present and future demand for modular and scalable communication architectures of many-core system, NOC has emerged as a viable alternative [13]. In the NOC, communication between the cores happens using an underlying fabric of routers connected in some of the network topologies. Each core is connected to a router and conventional data signal exchanges between cores is replaced by message passing through network router fabric. Using a network to replace the global wiring has an advantage of simplified structure, performance, and modularity. In the CMP having larger core count, design of NOC architecture mainly consist of inter-connection in between the cores, routers, memory associated with the cores, and topology that is used to connect different routers of the network.

There have been many research that studied the NOC architecture for the CMP system [56, 78, 73]. The two dimensional (2D) mesh based interconnect topology is the commonly used interconnection network technique due to its scalability and simplicity for the CMPs. Many real CMP systems, including Intel's Xeon Phi Processor [54] and Tilera's TILE64 [12], are based on mesh interconnection network. Fig. 1.1 shows

TH-2297_136201001



(a) TILE64 block diagram [12]



(b) Intel Xeon Phi block diagram [54]

Figure 1.1: Example of 2D-mesh based industry level CMPs

the examples of 2D-mesh interconnect based commercial CMPs. TILE64 has 64 tiles or cores connected using 8×8 2D-mesh NOC. While Xeon Phi processor has 36 tiles (each tile with 2 associated cores) and tiles are connected using 6×6 2D-mesh NOC.

1.1.4 Hybrid Network-On-Chip

In the many-core era, the performance and power consumption of the multi-core processors is limited by commonly used electrical NOC. Connor *et al.* in [86], explored that electrical NOC alone can not satisfy the power and performance demands of

TH-2297_136201001

the future CMP systems. Therefore, conventional electrical NOCs need to be either augmented or even replaced by the advanced network interconnects. Nowadays, many on-chip interconnect network technology, including optical NOC and wireless NOC is under exploration by the researchers [35, 123, 61].

Optical on-chip interconnect either supplement the electrical interconnection network or replaces it entirely. In [113], authors have explained that the energy and delay have significant gaps for the shrinking transistors size. Also, the authors have explained that the signals on electrical wires can be made to travel faster by inserting repeaters; this measure considerably increases the energy for data transmission. Therefore, the latency of the electrical interconnects is limited by the power budget and is likely to prohibit further performance and power scaling of the chip multiprocessor system, having a larger number of cores. Recently, many hybrid interconnection topology designs have been proposed by the researchers [114, 43, 113], which utilizes the 2D-mesh based electrical interconnects along with the optical interconnects.

1.2 Multi-threaded Application and Run-time Phase-wise Behavior

Technology advancement leads to the development of the multi-threaded and multi-process program models, that are designed to take advantage of the multiple processors and improve the overall performance; and also these program models take advantage of the multiple cores in the CMP system [87]. In the most popular user-level multi-threaded program model, the programmer maintains the user-level threads. User-level multi-threaded programming systems such as POSIX threads [22] and Win32 threads [92] have been developed to create and manage the user-level threads. Multi-threaded applications are being developed to take the benefits of the user-level threads and utilize CMP architectural abilities. In multi-threaded workload, a single application is sub-divided into threads based on its specific operations, and each thread can run in parallel on different cores of CMP. Data sharing in between the threads are more because they belong to the same application and uses the shared address space. In the modern-day scenario, many applications have been developed and are getting developed using the multi-threaded programming model. PARSEC [14] and SPLASH-2 [118] are examples of the benchmark suits that are produced using the multi-threaded programming model. Also, applications that are developed

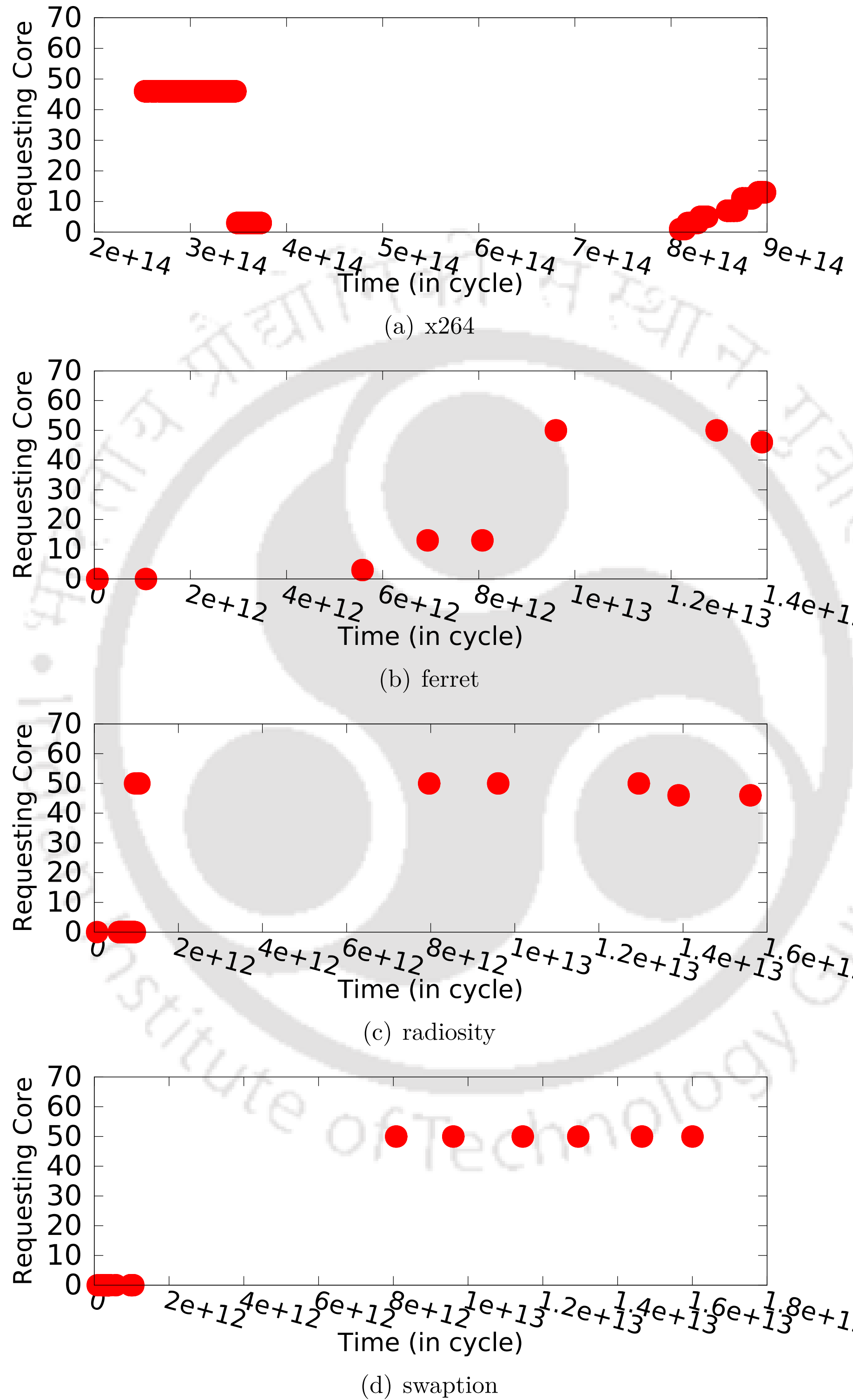
TH-2297_136201001

using OpenMP, Cilk, and Pthread programming tools are examples of multi-threaded workloads. A CMP remains underutilized if its application is not adequately disintegrated into multiple threads. Therefore, CMP performance may depend on the amount and characteristics of the parallelism in the applications.

There have been many studies that revealed the phase-wise behavior of most of the applications at the run-time [105, 104, 29]. Also, researchers have developed the tools and methods to analyze the phases of execution of the applications. Sherwood *et al.* in [105], shown that applications have phase-based behavior over many hardware metrics, for example, cache behavior, memory page access behavior, branch prediction, instruction per cycle (IPC), etc. In this context, we say that the page access behavior is also a characteristic which may change over the execution phases of the application. For example, a page that is extensively being used by a core in an execution phase may be needed by another core in another phase for the processing. Fig. 1.2(a), 1.2(b), 1.2(c), and 1.2(d) shows the access request pattern generated from the different cores at the run-time for the randomly selected data pages taken from the x264, ferret, radiosity, and swaption benchmarks respectively. For example, in the first graph (a) of the Fig. 1.2, the selected page is heavily used by the 54th core of the 8×8 two-dimensional mesh-based CMP for the duration between 2.5×10^{14} and 3.4×10^{14} time cycle of the execution. Further, the same page is needed heavily by the 2nd core (in the same 8×8 2D-mesh) between time cycle 3.4×10^{14} and 3.8×10^{14}. Again, the same page is needed by further different cores after 8×10^{14} time cycle. Therefore, for a CMP system having multiple memory modules (maybe hybrid memory modules) and memory controllers, many pages associated with the running application may need to migrate from one memory module to other at the run-time such that requesting core can access the data from its nearest memory module.

Based on the phase-wise behavior, the run-time of an application can be divided into multiple phases (or epochs). In [105], Sherwood *et al.* have used a fixed size phase length to make a balance between having a high capture rate and reducing the percentage of false positives. Contrarily, Shen *et al.* in [104] have used an off-line phase detection mechanism based variable length phase size. Therefore, for simplicity and to avoid off-line application processing, we use a fixed phase (or epoch) size throughout the thesis while considering the phase-wise behavior.

TH-2297_136201001



(a) x264

(b) ferret

(c) radiosity

(d) swaption

Figure 1.2: Access requests generated from different cores at the application run-time for the randomly selected pages.

TH-2297_136201001

1.3 Motivation

Nowadays, there is a huge increase in the complexity of the applications, and they require different CMP configurations to get the best performance. Modern CMP system consists of either homogeneous or heterogeneous cores or combination of both on the same chip. Also, once the chip is fabricated, we need to use it efficiently, so we require an efficient methodology to use the CMP in a better-way. Therefore, to utilize the architectural specifications of the CMP system and get the best possible performance, we need to perform the application mapping on to the CMP system.

In general, the affinity of a task to core gets considered in heterogeneous CMP environment where the task mapping to different core results in different execution time. This kind of work is well studied in [9, 19]. In the scheduling of tasks onto homogeneous multi-core CMP, scheduler either minimizes communication overhead between tasks [31, 108] or maps the tasks to the cores such that it can take benefit of data available at that core [77, 21, 37].

Formerly, for the CMPs with smaller core count (<10), application mapping on to CMP is recognized as a key research area to improve the overall performance and reduce the power consumption of the system. Moreover, for these CMPs, application mapping mainly considered the thread to core mapping, and not the data to memory mapping (as on-chip memory was smaller, in MBs). However, for the current and future CMP system where core count is huge (>100), researchers have proposed many future generation architectural designs such as high bandwidth and high density (in GBs) on-chip memory, and efficient interconnection network to satisfy the growing need of the CMP systems. Therefore, in this thesis, we investigate the impact of the application mapping, mainly due to the data mapping for the future generation CMP systems with higher core count.

Nowadays, researches have proposed the three dimensional (3D) stacking of the on-chip memory to provide the high bandwidth memory for the CMP systems. Though 3D-stacked on-chip memory has the capacity in tens of GBs that can fulfill the on-chip memory bandwidth requirement of the current as well as future CMPs system, it suffers due to its current cache like architecture [75]. In CMP, 3D-stacked memories are proposed to be architect either (a) to cache both local and remote data, or (b) to cache only the local data [69, 107, 30, 64, 124, 126, 96]. Caching only local data into the 3D-stacked memories enforces the CMPs to suffer inter-node latency overhead while accessing remote data. However, caching both local and remote data

TH-2297_136201001

onto the 3D-stacked memory requires a large coherence directory (about 64MB of coherence directory for 1GB memory) to ensure correctness [36, 32, 30]. Also, many on-chip interconnect network technology, including wireless NOC and optical NOC is under exploration by the researchers to provide efficient interconnection [35, 123, 61].

Therefore, considering the different NOC organizations and 3D-stacked memory for the CMP, and investigating the impact of the data placement becomes crucial for the current as well as future research. Sanchez *et al.* in [102], identified that latency overhead due to the network interconnect of the CMP is responsible for 60% to 75% of the miss latency. Also, miss latency after the last level cache (LLC) depends on the memory access time. Moreover, the miss latency increases with the increase in the NOC size as well as memory size. Therefore, hybrid NOC and on-chip 3D-stacked memory play a vital role to reduce the average memory access time.

The power consumption due to the NOC is very substantial in chip-multiprocessor systems and an important aspect for the optimization [130, 34, 38]. Mutlu *et al.* in [38], reported that NOC consumes about 40% of on-chip power. So, if we reduce the effective on-chip network traversal through the NOC, it reduces the on-chip power consumption significantly, which makes it another essential motivation to solve this problem. Furthermore, the use of the DRAM and PCM based hybrid memory can also reduce the system power consumption by reducing the power consumption associated with the DRAM refresh.

The latency overhead, as well as on-chip power consumption due to the interconnection network (or NOC), can be minimized by turning remote access into local. That can be achieved either by, (a) **caching the memory pages at local places and maintaining coherence**, or (b) **by moving the memory pages to local places judiciously**. Therefore, to avoid **the remote access overheads** as well as **the overheads associated with the large coherence directory** in the 3D-stacked memory-based CMP system, we performed the efficient page mapping on to the memory slices of the 3D-stacked memory. Further, this thesis investigates the effects of our proposed page mapping on to the performance, power consumption, and chip-area considering the CMP with hybrid NOC and hybrid 3D-stacked memory.

TH-2297_136201001

1.4 Objectives

The principal aim of this dissertation is to propose application mapping on to the 3D-stacked memory of the CMP system. In particular, the objectives are to efficiently map the applications on to the CMP considering the different current as well as future architectural design variations of the CMP system. Specifically, we consider the following current as well as future CMP architecture designs in terms of memory and NOC.

- **3D-stacked memory:** In this architectural design, we consider a large main memory which is placed on top of the processor layer in a 3D-stacked manner. This main memory can be of type- DRAM only, DRAM along with SRAM buffer, DRAM as a cache, DRAM as a cache along with the SRAM buffer, and DRAM-PCM based hybrid memory.

- **NOC:** In this architectural design, we consider an on-chip interconnection network between the cores of the CMP which is either a 2D-mesh organization of the electrical interconnects (termed as 2D-mesh NOC), or a combination of electrical as well as optical interconnects (termed as hybrid NOC).

Therefore, comprehensively, our considered CMP system is composed of (a) multiple cores that are connected using NOC, and (b) a large capacity 3D-stacked memory which comprises multiple distributed memory slices (or memory banks). Moreover, our objective is to consider above mentioned modern CMP design architectures and analyze the effects of the higher granularity efficient mapping techniques on to the system performance metrics. In higher granularity mapping techniques, we aim to map threads and pages associated with the applications on to the cores and memory slices of the CMP system, respectively. The following subsections describe the objectives of this thesis in brief.

1.4.1 Static Profile Based Mapping

In the static profile based mapping, our main aim is to generate a good page to memory slice mapping and thread to core mapping (based on simulated annealing and ant colony based genetic algorithms) such that the overall communication through the interconnection network can be minimized. We assume that profiling data is given

TH-2297_136201001

or generated by running the application once at the CMP system. Further, we use the profiled data and aim to optimize the following performance metrics.

1. Communication between threads (in terms of the number of communication between threads and distance between the cores having associated mapped threads).

2. Communication due to the memory accesses (in terms of the number of page access and distance between the cores having associated mapped thread and DRAM memory slice).

Mainly, we may need to analyze the effects of the proposed virtual mapping techniques, which can be used to reduce the overall on-chip communication (specifically related to the memory access) due to the remote data (or page) accesses for the 3D-stacked DRAM-based CMP system. Therefore, application mapping allows the large size 3D-stacked DRAM memory of the CMP system to be used in a different alternative way and without having the overheads associated with the coherence maintenance.

1.4.2 Run-time Dynamic Mapping

Static profile based mapping is not always useful as it needs the profiling statistics of the application execution, and therefore, we need to run the application at least once to get the statistics. Further, as most of the applications are having phase-wise behavior during their execution and from static profile based mapping, we found that the communication due to the page mapping dominates the thread to thread communication. Therefore, our objective is to design methodology for a hardware-based dynamic run-time page mapping on to the memory slices to minimize the overall communication.

Further, we aim to extend the hardware-based dynamic run-time virtual page mapping and compare the overall execution time with the state-of-art work where 3D-stacked DRAM is used as coherence cache. In this comparison, we may identify that for the CMP having larger size 3D-stacked DRAM memory, whether our extended version of the dynamic run-time virtual page mapping performs better as compared to the 3D-stacked DRAM used as coherence cache (as proposed by state-of-art work in [30]).

TH-2297_136201001

1.4.3 Run-time Mapping Considering Hybrid Memory

3D-stacked DRAM memory having through-silicon-vias (TSVs) provides a higher bandwidth on-chip memory as well as a wide range of design flexibility to the CMP systems. Although the 3D-stacked DRAM has various advantages, its power consumption is rapidly growing with the capacity increase. Researchers have proposed the large size 3D-stacked DRAM-PCM based hybrid memory that fulfills the on-chip memory requirements of the future CMP system having a larger core count. However, for the 3D-stacked DRAM-PCM based hybrid memory, both memory organizations, either (1) proposed by us (simple memory without allowing to store multiple copies and virtual page mapping) or (2) memory as coherence cache suffers because of the massive number of memory pages.

Therefore, the next objective of this thesis is to consider the CMP system having DRAM-PCM based hybrid memory and propose a new access aware dynamic virtual page mapping technique. For, CMP system having DRAM-PCM based hybrid memory, we aim to analyze the feasibility of our proposed technique in reducing, (1) the energy consumption by avoiding the DRAM refreshes and (2) the overall execution time by reducing the remote page access.

1.4.4 Performance Analysis of CMP having 3D-stacked DRAM and Hybrid NOC

The number of cores on CMPs are growing at an exponential rate to increase the performance. However, inadequate on-chip interconnection and memory bandwidth have diminished the potential of these CMPs. High performance interconnects, 3D-stacked main memory, and large on-chip caches are the architectural parameters used to tackle these issue. For a fixed die-size, high performance interconnects, and 3D-stacked memory fosters the growing rate of the cores on a CMP whereas increasing the size of on-chip cache poses a restriction.

Therefore, in this part of the thesis, we aim to study the trade-off between the performance and overall chip area (evaluated using the number of cores, types of core and cache area per core) of the 3D-stacked DRAM-based CMP having hybrid NOC. In this last objective, we aim to analyze the effects of the adaptive virtual page mapping on to the instruction per cycle (IPC) and communication of the 3D-stacked DRAM-based CMP having hybrid NOC.

TH-2297_136201001

1.5 Contributions

The major contributions of the thesis are described in the following subsections.

1.5.1 Static Profile Based Mapping

In this part of the thesis, we use a profile-driven application mapping of the applications to minimize the on-chip communication of the 3D-stacked DRAM-based CMP. In profile based mapping, we run the application on target CMP system architecture and get the statistics about the thread to a thread communication, and thread to virtual page access count. Moreover, we use the profiled information to optimize the overall on-chip communication of the application for the next run.

In this part, we describe (a) simulated annealing based thread only mapping onto 3D-stacked memory, (b) virtual page only mapping of multi-threaded application onto 3D-stacked memory, (c) virtual page mapping followed by thread mapping onto 3D-stacked memory, and finally (d) combined thread and virtual page mapping onto 3D-stacked memory. An essential point to note here is that we have used simulated annealing based thread mapping for the simplicity, further for comparison purpose we have used a recent ant colony heuristic-based thread to core mapping. Moreover, we have proposed virtual page mapping techniques and two others that uses the simulated annealing based thread to core mapping. The brief detail of these techniques is given as follows:

1. Thread mapping: In this case, we try to map threads of the applications to the core to minimize overall thread to thread communication of the system. To perform the thread to core mapping, we have used a simulated annealing method and further compared the result with a recent genetic ant colony based method [41].

2. Virtual page mapping: In this mapping process, virtual pages of the application get mapped to physical pages in such a way that it reduces the overall communication involved in all the virtual page accesses from all the threads. In this case, virtual pages get mapped to the physical page of a DRAM slice (and ultimately to one DRAM controller which is attached to the communication network of CMP).

TH-2297_136201001

3. Thread mapping followed by virtual page mapping: In this case, we generate a good thread to core mapping using some method (simulated annealing is used), and after that virtual page to DRAM memory slice mapping is performed considering this new thread to core mapping.

4. Combined thread mapping and virtual page mapping simultaneously: In this approach, we look at both virtual page to DRAM slice mapping and thread to core mapping simultaneously, so that the mapping take advantage of cost-effective options from both thread communications and memory accesses. Both thread communication and memory access get prioritized equally.

After evaluation using all these above options, we choose the minimum cost mapping option for the next run of the application to minimize the overall on-chip communication.

1.5.2 Self-adaptive Run-time Page Mapping

As most of the applications exhibit phase-wise behavior at the run-time [29]; therefore, core to core communications and virtual page access pattern (or footprint) of an execution phase associated to an application may be different with another phase of the same execution. Therefore, a static mapping may not be suitable for the complete duration of execution of an application. Also, in the static profile based mapping, we need to collect the statistics by running the application once and then only we can apply the static profile based mapping. However, in the case of dynamic run-time mapping, we do in the first run itself or mapping changes as the application run-time in progress. So, we designed an approach to map the application at the run-time for different phases of the same application.

From the results of the profile based static mapping (as stated in Sub-section 1.5.1), we found that the overall communication reduction due to virtual page mapping is significantly higher as compared to thread mapping. Also, as described in [47], the overheads associated with destroying a thread, transferring thread state (consist of a program counter, a set of registers, and a stack of procedure records containing variables local to each thread), creating a new thread and initiating remote execution make runt-time thread migration relatively tricky. Hence, for run-time mapping and remapping, we considered the mapping of the virtual page to DRAM memory slices only and ignored the thread to core mapping.

TH-2297_136201001

In the earlier profile based static virtual page to DRAM memory slice mapping, a virtual page gets associated with a DRAM slice for whole execution period of the application. However, in the case of dynamic run-time page mapping, we do in the first run itself, or page gets mapped and remap as it run-time progress. A virtual page may migrate from one DRAM slice to other DRAM slices.

Recently, many research proposals studied the use of 3D-stacked DRAM as on-chip cache. Specifically, in [107], CMP system uses 3D-stacked DRAM cache to store only the local data. In such a multiprocessor system, a data can reside in only one DRAM cache, which eliminates the need for coherence support for such DRAM cache. For remote side data, this system has to rely on small on-die caches (such as L2 caches). So, any on-die cache miss to the remote data leads to significant latency overhead due to inter-node traversal.

Further, Chou *et al.* in [30] has recently proposed another DRAM cache architecture for the CMP system. Their proposed DRAM cache architecture allows the caching of both the local data and remote data at the cache block-level granularity. Also, their DRAM cache is designed to ensure the correctness by employing coherence support. They placed the coherence directory in the 3D-stacked DRAM (termed as embedded coherence directory) and reused the existing SRAM based on-chip cache directory to cache the recently accessed embedded coherence directory entry. However, using DRAM as a coherent cache for the CMP system (with the number of cores inside the chip is in the range ≥ 100) requires the large size of the coherence directory, latency overhead to access coherence directory, tag storage and lots of overhead to maintain the correctness.

Therefore, this part of our thesis considers the adaptive run-time page (data) mapping similar to as stated in Sub-section 1.5.2 and squarely focuses on a comparative study with a coherent and non-coherent DRAM cache.

1.5.3 Run-time Page Mapping Considering Hybrid Memory

Many researchers have explored 3D-stacked hybrid memory architect using PCM and DRAM as an alternative to only PCM or DRAM memory [126, 96, 95]. Typically, these hybrid memories regulate the placement of their data (or pages) to minimize the leakage power of the DRAM memory and the high access latency of the PCM memory. In a recent survey [127], authors have explored that studies related to the hybrid 3D-stacked memory (or hybrid cache architecture) have mainly focused on

TH-2297_136201001

to the data migration between the parts of a hybrid memory module. Also, the survey shows that studies related to the hybrid memory system have considered the sophisticated non-uniform cache architecture schemes.

Therefore, in this contribution, we have considered non-coherent 3D-stacked hybrid DRAM-PCM memory-based CMP system; and proposed an access-aware self-adaptive run-time page (or data) mapping. Mainly, in this contribution, we have used a simple DRAM access-aware page placement technique between DRAM and PCM for the hybrid memory slice to reduce the DRAM refresh operations and its associated power consumption overhead. Further, we performed an access-aware self-adaptive page mapping for the optimized page placement between the different hybrid memory modules of the 3D-stacked hybrid memory.

1.5.4 Performance Analysis of CMP having 3D-stacked DRAM and Hybrid NOC

The number of cores on a CMP is growing at an exponential rate to increase the performance. However, inadequate on-chip interconnection and memory bandwidth have diminished the potential the CMP. High performance interconnects, 3D-stacked main memory, and large on-chip caches are the architectural parameters used to tackle these issue. For a fixed die-size, high performance interconnects, and 3D-stacked memory fosters the growing rate of the cores on a CMP whereas increasing the size of on-chip cache poses a restriction.

In this contribution, we consider a self-adaptive data page mapping onto a 3D-stacked DRAM-based CMP with hybrid interconnection network. Additionally, we study the trade-off between the performance and cache size per core using different combinations of the interconnection network, 3D-stacked DRAM memory (on-chip or off-chip) and an adaptive data page mapping. This part of thesis, analyzes the effect of reduced cache size on to the system performance while considering the benefits of, (1) 3D-stacked DRAM, to provide high memory bandwidth, (2) high performance optical interconnect, to enable low-latency communication and efficient memory utilization, and (3) a self-adaptive run-time page mapping similar to as explained in Sub-section 1.5.2.

TH-2297_136201001

1.6 Thesis Organization

This thesis comprises seven chapters.The chapter wise organization of the thesis is given as follows:

1. **Chapter-1:** This chapter provides the introduction and motivation along with the objectives and contributions behind the research work.

2. **Chapter-2:** This chapter presents a survey of the previous related works that are needed to get the idea of the state-of-art works.

3. **Chapter-3:** This chapter presents an overview of the system and applications models that are used to perform the contributions of the thesis.

4. **Chapter-4:** This chapter presents the static profile based mapping approaches on to the 3D-stacked DRAM based CMP considering the simulated annealing based thread to core mapping. The contents of this chapter have been published in [88].

5. **Chapter-5:** This chapter presents the dynamic run-time virtual page mapping approach on to the 3D-stacked DRAM based CMP. The mechanism proposed is compared with a recent state-of-art work [30]. The partial contents of this chapter are based on the published work reported in [88].

6. **Chapter-6:** This chapter presents the dynamic access-aware run-time mapping considering DRAM-PCM based hybrid memory. The chapter is based on the work published in [89].

7. **Chapter-7:** This chapter discusses the performance and cache area trade-off analysis considering 3D-stacked DRAM based CMP having hybrid NOC.

8. **Chapter-8:** This chapter discusses the conclusion arrived at, and the future research scopes related to this thesis.

TH-2297_136201001

2

Related Works

Previous works related to this thesis can be categorized into, (a) application mapping, (b) 3D-stacked DRAM memory, (c) 3D-stacked hybrid memory, (d) NOC, (e) hybrid NOC and its uses, and (f) area and performance trade-off implications using hybrid NOC and 3D-stacked memory. Following sections explain the related works associated with the above categories.

2.1 Application Mapping

In general, the affinity of the task to core gets considered in the heterogeneous CMP environment where the task mapping to different core results in different execution time for the task. This kind of work is well studied in [9, 19]. In [9], Balakrishnan *et al.* have presented a detailed study to describe the behavior of commercial applications running on multi-core systems where each core have different performance capabilities. They observed that performance asymmetry in each core could have an unintended negative impact on applications; therefore, it becomes difficult to predict their performance. So, they suggested robust application designs that can adjust the applications dynamically for the varying compute capabilities of the system.

Peter Brucker in his book [19], has covered many recent as well as classical scheduling problems for single and parallel machines. From the theoretical perspective, he discussed and classified the task scheduling problems for both preemptive and no-preemptive versions of the task scheduling. Methods to solve these problems such as linear programming, branch-and-bound algorithms, dynamic programming,

TH-2297_136201001

and local search heuristics are also summarized in this book. However, the book does not consider the data mapping and interconnect related issues for these problems.

In the scheduling of tasks onto multi-core CMP, scheduler either minimizes communication overhead between tasks [31, 108] or maps the tasks to the cores such that it can take benefit of data available at that core [77, 37]. In [31], Chou *et al.* have addressed the run-time task allocation problem for the embedded NOC platform having heterogeneous processing resources. They have proposed efficient algorithms to minimize internal and external contention and communication costs. Also, they characterized the incoming applications by the application characterization graph (represented as directed graphs) and mapped to a region in the mesh which contains several cores and minimized both internal and external communication contention. Sreepathi *et al.* in [108], have presented many task mapping algorithms that combine the insights from the network topological information with the application's behavior together to provide an efficient task assignment.

In [59], Kwok *et al.* have presented an extensive survey of the algorithms used for the static scheduling problem. They have introduced the target multiprocessor model as a network of processing elements, each of which comprises a processor and a local memory unit and communication between them is achieved solely by the message passing. They described the objective of the scheduling algorithms to minimize the schedule length by adequately allocating the atomic program task (represented as a directed cyclic graph) to the processing elements. Also, their objective includes sequence maintenance between the start times of different program tasks such that the precedence constraints are preserved. However, scheduling problems in this survey does not consider the issues related to future multiprocessor designs and memory mapping.

In [77], authors have presented a run-time system assisted data distribution scheme. Their method allows the programmers to control data distribution in a portable fashion, without forcing the programmers to understand the low-level system details. Also, their scheme requires nearly the same programmer efforts as regular calls to the malloc. In [37], authors have presented a cache hierarchy-aware code mapping and scheduling strategy for multi-core systems. Their mapping strategy considers the application data access patterns and on-chip cache hierarchy and determines a schedule for the iterations assigned to each core to get the dependency-free loop nests.

Application tasks, data placement, and scheduling in multiprocessor have been

TH-2297_136201001

studied by Suhendra *et al.* in [109]. In [109], they proposed an integer linear program formulation to do task scheduling, scratchpad memory partitioning, and data allocation to partitioned scratchpad memory to improve the performance. In [27], Chen *et al.* have proposed an approach to map task and data of the data-centric application (which is a unique kind of benchmark) to chip-multiprocessor. However, local memory capacity they considered is small (64 KB/Processor) and in their design, they argue that placing large memory on the logic layer is not feasible. In a recent study [106], Singh *et al.* have surveyed different resource allocation or mapping techniques for multi-core systems. They have identified different upcoming trends and challenges based on the comparative study.

2.2 3D-stacked DRAM Memory

Evolution in the chip manufacturing technology enabled the placement of the DRAM memory onto the chip as another 3D-stacked layer [20, 62, 57, 28, 48, 67, 69, 117]. The capacity of the 3D-stacked DRAM memory is in the range of gigabytes (GBs), which makes it capable of holding a larger number of data sets. For example, Intel Knight Landing multiprocessor is equipped with 16GB of high bandwidth 3D-stacked DRAM [107].

Researches have proposed to use the 3D-stacked DRAM either as cache or as on-chip main memory because of its higher bandwidth, smaller cost and design complexity [67, 69, 117]. In [67], Liu *et al.* have examined how 3D IC fabrication technology can improve interactions between the processor and memory. They have shown that bringing main memory on-chip gives us a near-perfect performance by reducing the memory access latency. In [69], Loh *et al.* have proposed a different 3D-stacked DRAM memory design approach to fully exploit the 3D integration technology. They have explored a more aggressive 3D-stacked DRAM organization that makes better use of the die-to-die bandwidth provided by 3D stacking. Also, in [69], authors have revealed that revisiting the memory system organization in a 3D context can provide much more performance improvements.

Furthermore, in [117], Woo *et al.* have studied the 3D-stacked memory architecture and re-designed the L2 cache and its interface to the 3D-stacked DRAM. Their technique SMART-3D improves the latency by exploiting the high density and bandwidth of TSV between the last-level cache, 3D-stacked DRAM, and processor. In

TH-2297_136201001

[25], Chandrasekar *et al.* have proposed system and circuit level power modeling of 3D-stacked wide I/O DRAMs. Their model for the 3D-stacked wide I/O DRAM memories is almost as accurate as detailed circuit-level power models of the 3D-stacked DRAMs.

Lee *et al.* in [62], have suggested an approach to improve the 3D-stacked memory bandwidth at low cost by simultaneous multi-layer access in, thus making better use of the bandwidth that TSV offers. Further, to avoid channel contention, they have proposed approaches for the coordination between the layers while accessing the data simultaneously. In [40], authors have proposed an architecture to place the main memory on top of the processor using 3D fabrication technologies. Their proposed memory design takes the benefits of the 3D architecture and bridges the performance gap incurred due to the limited off-chip main memory bandwidth and access latency.

In the article [127], authors have presented that the researches related to the 3D-stacked memory system have considered the sophisticated non-uniform cache architecture schemes. However, 3D-stacked memories suffer due to the coherence related issues for larger size coherence directory associated with these memories. Therefore, it is an urgency to look upon the newer dimensions of memory management and organization for the 3D-stacked memories. In [107], authors use the 3D-stacked DRAM memory as a cache for their considered CMP system model, and this memory stores only local data within it. For remote side data, the system needs to rely on small on-die caches. Therefore, each on-die cache miss suffers due to the inter-node traversal overheads while accessing remote data. In another recent work [30], authors have proposed a DRAM cache architecture that allows caching both the local data and the remote data at the cache block-level granularity. They placed the coherence directory in the 3D-stacked DRAM and used an SRAM based on-chip buffer to cache recently accessed coherence directory entries.

Dynamic run-time page mapping of virtual pages to 3D-stacked memory and considering the on-chip 3D-stacked memory for physical pages may provide a better solution to improve the performance. Therefore, it is essential to know the efficient organization of the translation lookaside buffer (TLB). In [72], authors have used inter-core cooperative prefetchers and shared last-Level TLBs for TLB usage improvement in CMP. Further, the larger size of main memory requires a corresponding increase in the processor's TLB resources to avoid performance bottleneck. In [91], Pham *et al.* have presented a multi-granular TLB organization that significantly in-

TH-2297_136201001

creases its effective reach. Also, their method reduces the miss rates substantially and requires no additional OS support.

2.3 3D-stacked Hybrid Memory

Although the 3D-stacked DRAM has various advantages, its power consumption is rapidly growing with the capacity increase. DRAM memory consumes 20% to 40% of the overall system power [66, 111]. To tackle the DRAM power overheads associated with the larger size DRAM, phase change memory (PCM) has been explored as an alternative to the DRAM [129, 99]. PCM has better scalability (higher density), non-volatility, and lower leakage energy as compared to the DRAM memory. Despite many advantages, PCM has some drawbacks as compared to the DRAM memory, for example, higher read/write latency and lower write endurance. Typically, when compared with the DRAM memory, each read and write operation associated to the PCM memory has about 4-6× and 6-32× longer latency, respectively; also, each read and write of the PCM memory consumes 2× and 10-140× more energy as compared to the DRAM memory, respectively [124, 93]. Further, the endurance value of PCM memory is about 10^8 which is far lower than the DRAM endurance value 10^{15} [63].

Therefore, hybrid memory constituted using DRAM and PCM has been proposed as a potential solution to take the advantages of the DRAM and PCM memory components [124, 93, 64]. In [124], authors have developed a row buffer locality-aware hybrid memory caching policy to utilize the benefits of the DRAM as well as PCM memory components of the hybrid memory. In this paper, authors have observed that PCM array access latency is much higher as compared to the DRAM array access latency. Therefore, they developed a caching policy that keeps track of the rows that have high row buffer miss counts and places only such rows in the DRAM memory component.

In [93], Pourshirazi *et al.* have proposed a technique to eliminate the DRAM refresh operations in the DRAM-PCM based hybrid memory. Moreover, their method evicts the non-accessed rows from DRAM memory if it is time to refresh those DRAM rows. In [64], Lee *et al.* have studied the effects of various DRAM and PCM memory configurations on the system performance and energy consumption considering the DRAM-PCM based hybrid memory. Further, authors have proposed a novel DRAM cache design to use in conjunction with the PCM memory such that the performance

TH-2297_136201001

and energy efficiency of the hybrid memory can be maximized.

Nowadays, 3D-stacked hybrid memory is becoming popular due to its high capacity and scalability benefits. Many recent studies, including [94, 39, 127], have studied the 3D-stacked hybrid memory architect using DRAM and non-volatile (including PCM, MRAM, etc.) memory technologies. In [39], Fawibe *et al.* have presented the idea related to the address space organization into pages and virtual address to physical address translation considering the hybrid 3D-stacked DRAM and PCM memory. In [94], Zhao *et al.* have introduced different types of heterogeneous multi-core architectures considering the emerging 3D-stacked DRAM and non-volatile memory technologies. Moreover, Zhao *et al.* have demonstrated the potential benefits towards future application requirement and advantages of leveraging 3D integration on heterogeneous architectures.

Some studies related to the hybrid memories considering the scratch pad memory and cache memory are also being performed [80, 119]. However, these memories are smaller in size as compared to the 3D-stacked hybrid memories. Therefore, coherence maintenance for these hybrid memories is easier due to the smaller coherence directories.

2.4 Network-On-Chip

Initially when up to tens of cores was there on a single chip, the traditional bus-based interconnect network performed well. However, in the many-core era, as the number of cores residing on a single chip is increasing rapidly, the traditional bus-based network cannot handle the synchronization problems of these large systems. Use of NOC in place of ad-hoc global wiring facilitates the modular design to connect the components of the chip. In NOC, communication between functional modules happens using an underlying fabric of routers connected in any of the network topologies. The structured network or NOC provides a well-controlled electrical parameter that enables the use of high-performance circuits to reduce the latency and increases the network bandwidth [33]. In [13], authors have studied the NOC and found that for present and future many-core systems, it is a viable alternative to provide modular and scalable communication architecture.

There have been many studies that proposed different types of the NOC designs; for example ring, tree, butterfly, 2D-mesh, hyper-cubes, torus, etc. [26, 58, 16, 53].

TH-2297_136201001

In [101, 15], the comparison between the state-of-the-art works related to the NOC design, how NOCs are being evaluated, which aspects have been covered till now, and the area which needs more research effort have been studied in detail.

Among all the NOC designs, 2D-mesh is the best-suited network topology for the CMP systems because of its simplicity and scalability. In addition to the advantages, 2D-mesh NOC has some disadvantages such as long network diameter and energy inefficiency because of the extra hops caused due to the long network diameter [98]. Moreover, Reshadi *et al.* in [98], have proposed routing algorithm termed as extended XY to improve the efficiency of the 2D-mesh NOC. Many real CMP systems, including Intel Knights Landing [107] and Tilera64 [12], are based on the 2D-mesh NOC.

2.5 Hybrid Network-On-Chip

As the trend toward many-core processors continues to grow, the on-chip communication fabric (networks-on-chip) has become a limiting factor regarding performance and power consumption. In survey article [113], authors have explained that the energy and delay have significant gaps for the shrinking transistors size. In the same survey article [113], authors have suggested that the signals on electrical wires can be made to travel faster by inserting repeaters, and this measure considerably increases the energy for data transmission. Therefore, the latency of the electrical interconnects is limited by the power budget and is likely to prohibit further performance and power scaling of the chip multiprocessor system, having a more significant number of cores. The survey also shows that silicon photonics is widely considered as one of the most groundbreaking technologies that allow ongoing performance and power scaling in chip design. The silicon photonic is having a transformative effect on the way chips are designed from rack-scale computing down to many-core. In the survey article [113], authors have provided an exhaustive study on the research efforts that have been conducted in the realm of on-chip optical interconnects.

Shacham *et al.* in [103], proposed a circuit-switched photonic NOC that provides enormous transmission bandwidth while consuming minimal power. Moreover, authors have also covered some critical design issues such as topology, path-setup procedures, routing algorithms, and deadlock avoidance rules along with the recovery procedures for the photonic NOC. Ye *et al.* in [123], have proposed an optimized 3-D mesh-based optical NOC. Their floor-plan follows the regular 3-D mesh topology;

TH-2297_136201001

however, all the optical routers are implemented in a single optical layer. Further, authors have compared the performance and energy efficiency between the 3D mesh-based optical NOC and the 2D mesh-based electronic NOC.

Werner *et al.* in [114], have proposed a hybrid network-on-chip topology that decreases the power consumption efficiently by combining electrical and optical links. Also, they identified that electrical links are best suited for the near distance communications and optical links are for longer distances. Bahirat *et al.* in [8], proposed a hybrid photonic NOC that uses a photonic ring based dedicated layer on top of the electrical 2D mesh-based NOC which reduces the power consumption by 13× as compared to the all-electrical 2D mesh NOC.

2.6 Area and Performance Trade-off Implication Using Hybrid NOC and 3D-stacked Memory

To support the growing need of the applications, the number of cores on a CMP is increasing at an exponential rate. As the number of cores on a CMP increases, the CMP system needs high bandwidth interconnection network along with the high capacity on-chip memory to give the higher performance [38, 102]. Advanced architectural designs such as large 3D-stacked memories and efficient high end interconnects have been proposed to foster the growing rate of the cores on a CMP. Also, the study has shown that for a fixed-size die, increasing the number of cores on a chip reduces the on-chip cache per core [7]. However, on-chip caches are one of the vital parameters to get better performance, and a shortage of caches may degrade the system performance. Therefore, we aim to analyze the trade-off between the performance and cache size per core for the CMP systems having 3D-stacked DRAM memory and hybrid interconnection network. Some, related previous studies have been done to study the performance and cache size trade-off, but none of them have used modern CMP system architecture as we have considered.

Authors in [7], have introduced an analytical model to study the trade-offs of utilizing increased chip area for the larger cache size versus more number of cores. Oh *et al.* in [82], have presented an analytical model to study the trade-off between the core count and the cache capacity in the CMP based on different cache organizations. Their presented model enables to quickly study the effects of the chip area allocation parameters on the system performance. In [71], authors have claimed that larger

TH-2297_136201001

cache reduces the die area available for the cores and performance degrades due to longer access latency. Moreover, for the data-center processors (termed as scale-out server processors), they have proposed a methodology to tightly couple a number of cores with comparatively smaller cache using the low latency interconnect and replicated this structure (termed as pod) to form an optimal system. Moreover, there is no direct connection between the pods, as these are stand-alone servers.

In [10], Balfour *et al.* have developed a detailed area and energy models for on-chip interconnection networks. Further, they have described the trade-offs while designing the efficient networks for tiled CMP system. They have also shown that the architectures commonly assumed in on-chip networks studies do not perform well in the CMPs if the core count increases. Huh *et al.* in [52], have compared performance and area trade-offs for the CMP implementations to determine the number of cores, in-order or out-of-order issues, and on-chip cache size for the future server CMPs.

Hill *et al.* in [49], have used Amdahl's Law for the CMP system and build a model to obtain speedups for asymmetric, symmetric, and dynamic CMP systems. Yavits *et al.* in [122], have developed an analytical solution to optimize the CMP cache hierarchy and optimally allocating the area among the hierarchy levels. Their proposed model allows performance optimization under typical CMP restrictions, such as constrained power budget, constrained area, limited off-chip memory bandwidth, and limited NOC capacity. In [121], authors have proposed an analytical solution to optimize 3D CMP cache hierarchy. Their method allows optimal partitioning of the cache hierarchy into 3D silicon layers and optimal allocation of the area among all the cache hierarchy levels such that area and power can be minimized.

TH-2297_136201001

TH-2297_136201001

3

System Model and Application Model

In this chapter, we explain the details of our considered CMP system model and its different variations along with the application model, which are used throughout the thesis. We cover all the essential details in this chapter, and it remains persistent throughout the thesis unless otherwise explicitly mentioned. Moreover, we explain them further in their respective chapters if required.

3.1 System Model and its Variations

The advent of three dimensional (3D) chip fabrication technology allows us to stack and fabricate different technological layers together over each other on a single chip [20, 112, 126]. It allows us to stack DRAM memory, optical interconnects, non-volatile memories (including phase change memory) on top of the chip.

Our considered CMP system architecture can be viewed as a 3D-stacked CMP having mainly two types of layers, and these are (a) processor layer and, (b) 3D-stacked memory layer. The **processor layer** comprises N processor cores $\{c_0$, c_1, c_2, $\cdots$, $c_{N-1}\}$ and these processor cores (or cores) are connected using an electrical $k_1 \times k_2$ 2D-mesh ($k_1 \times k_2$=N) network of routers present at this layer. All the cores at the processor layer are homogeneous. Each core has their private caches (L1-I, L1-D and L2, and cache coherence is maintained at L2 level using MESI protocol) and each core is associated with a router present in the 2D-mesh interconnection network.

TH-2297_136201001

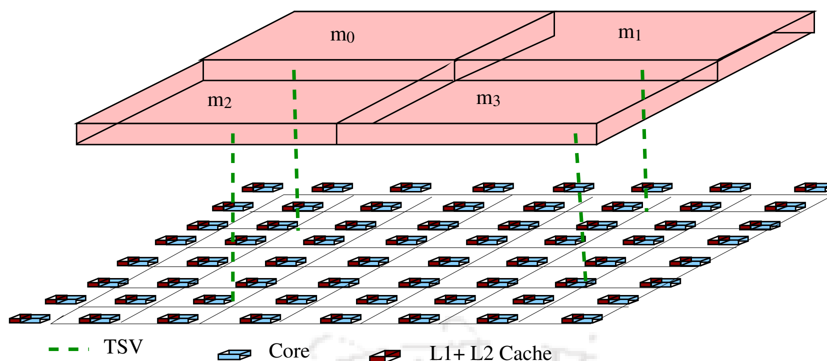


Figure 3.1: Example: 3D representation of the system model

3D-stacked memory layer, on top of the processor layer, is made of M $\{m_0, m_1, m_2, \cdots, m_{M-1}\}$ number of on-chip 3D-stacked memory slices and each slice is having a memory controller (MC) associated with it. These memory slices are assumed to be non-coherent, and they do not share data between them. Each **3D-stacked memory slice** may be multi-layered, and connection in between them as well as to the processor layer is made using TSVs. Moreover, there is some *specific router* positions in the 2D-mesh at the processor layer, where each 3D-stacked memory slice is connected using an additional router port and TSVs. The number of specific routers (associated to each memory slice) and the specific router positions at the processor layer can be derived or identified by using overall minimum Manhattan distance clustering principle [90] or geometric modeling principle [76]. However, we have used the minimum Manhattan distance clustering principle to decide the locations of the specific position routers and their numbers. The number of memory channels per MC can be calculated using the approach as given by Abts *et al.* [5]. However, in our case, we use one channel per MC for simplicity.

Fig. 3.1, shows an example of the 3D representation model associated with the considered CMP system architecture. In Fig. 3.1, processor layer has 64 cores that are connected using an 8×8 2D-mesh of electrical interconnects. The 3D-stacked memory layer is placed on top of the processor layer. 3D-stacked memory layer comprises four 3D-stacked memory slices m_0, m_1, m_2 and m_3, and these slices may be multi-layered.. Each 3D-stacked memory slice is connected to one of the processor layer router placed at a certain position, and that processor layer router is termed as a specific router.

Fig. 3.2 shows a simplified and detailed view of Fig. 3.1 associated to the 8×8 CMP system. The right part of Fig. 3.2, shows the four non-overlapping areas associated with the four 3D-stacked memory slices $\{m_0, m_1, m_2$ and $m_3\}$ for the 8×8

TH-2297_136201001

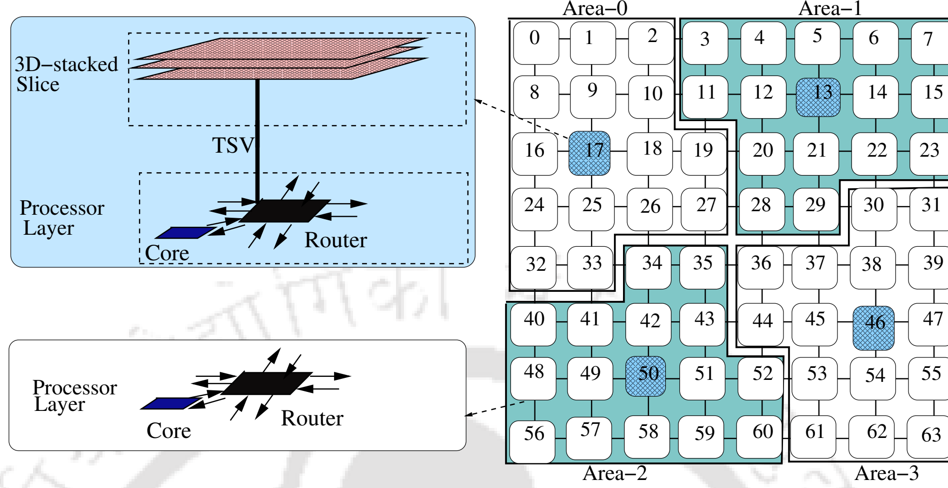


Figure 3.2: Detailed system model representation showing four non-overlapping areas.

CMP system. Positions of the 3D-stacked memory slices associated with the specific position processor layer routers prompt us to consider that every core belongs to a particular area covered by a 3D-stacked memory slice. We used a 1D-array entitled as core to slice table (CST), which gives the value of the 3D-stacked memory slice number $SNum$ for each core. The value $SNum$ is corresponding to the 3D-stacked memory slice number which is closely associated (or local) to the core indexed by 1D-array.

Therefore, each core belongs to a coverage area associated to a 3D-stacked memory slice, and this is the closest among all the 3D-stacked memory slices. The $CST[i]$ array value represent the closest 3D-stacked memory slice to the ith core. For an example, corresponding to the four 3D-stacked memory slices m_0, m_1, m_2 and m_3 the $SNum$ can have a value from 0, 1, 2 and 3. 3D-stacked memory slices m_0, m_1, m_2 and m_3 are connected to the processor layer routers placed at 17th, 13th, 50th and 46th positions respectively. So, the values of CST array can be derived as CST[64]={0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 3, 3, 0, 0, 2, 2, 3, 3, 3, 3, 2, 2, 2, 2, 3, 3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 3}. Therefore, based on the CST[64] array values, core indices can be divided in to four areas. The elements (or core indices) associated to the four areas are given as follows:

- **area-0**: {0, 1, 2, 8, 9, 10, 16, 17, 18, 19, 24, 25, 26, 27, 32, 33}.

TH-2297_136201001

- **area-1**: {3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29}.
- **area-2**: {34, 35, 40, 41, 42, 43, 48, 49, 50, 51, 52, 56, 57, 58, 59, 60}.
- **area-3**: {30, 31, 36, 37, 38, 39, 44, 45, 46, 47, 53, 54, 55, 61, 62, 63}.

Cores associated to area-0, area-1, area-2 and area-3 are in the coverage area of 3D-stacked memory slices m_0, m_1, m_2 and m_3 respectively. In this example, the maximum Manhattan distance between a core and its local 3D-stacked memory slice is 3.

Further, based on the objective of this thesis, our considered CMP system architecture can have any of the following four architectural components on top of the chip in a 3D-stacked manner.

- 3D-stacked DRAM memory, and
- 3D-stacked DRAM memory along with an SRAM buffer, both are similar to the architecture as described in [69].
- Combination of 3D-stacked DRAM and PCM memory, similar to the design as given in [126].
- Combination of 3D-stacked DRAM and optical interconnect, similar to as described in [69, 8, 61].

In traditional 2D architecture, off-chip memory access is limited by slow (order of $\approx$ MHz) off-chip buses [46]. Loi *et al.* in [70], have unveiled that the vertical on-chip buses (termed as TSV) in the 3D design have an access frequency in the order of $\approx$ GHz. Therefore, limitations of the off-chip buses (in case of off-chip main memory) are eliminated by using a 3D-stacked layer on top of the processor layer, that takes the benefits of the high-speed on-chip TSV. The cores and memory slices which are connected using TSV can be considered on-chip as reported in [67, 40]. Also, for the CMP having a larger core count, consider the smaller value of the chip height as compared to the length and breadth. Therefore, we can assume that communication time and distance from the 3D-stacked memory layer to processor layer routers using TSVs are negligible.

Therefore, based on the constituents of the CMP, our considered generic system model (as shown in Fig. 3.1) get converted into four distinct CMP system models that are used to achieve the objective of the thesis. Following sub-sections explain these four distinct CMP system models in detail.

TH-2297_136201001

3.1.1 DRAM Memory at the 3D-stacked Memory Layer

In this variation of the generic CMP system model, each 3D-stacked memory slice is having an on-chip memory controller (MC) associated with its 3D-stacked DRAM memory slice. Therefore, there are M $\{m_0,\ m_1,\ m_2,\ \cdots,\ m_{M-1}\}$ number of on-chip 3D-stacked DRAM memory slices and their MCs. The memory capacity of all the DRAM slices is equal. The considered CMP architecture is similar to as described in [74], where 3D-stacked DRAM memory layer is stacked on top of the processor layer. In general, to reduce the temperature of the chip, the memory slices are placed in the bottom layers, and the processor layer (logic layer) is the top layer of the 3D-stacked chip so that heavily utilized processing cores can dissipate heat easily [110]. Logically both the architectures are the same.

Fig. 3.3 shows an example of the CMP system model having 3D-stacked DRAM memory slices on top of the processor layer. This CMP system model is same as shown in Fig. 3.2, except the detailed representation and connection of the 3D-stacked memory slices. The left part (three-dimensional view) shows the 3D-stacked DRAM memory slice organization and its interconnection with the processor layer routers using TSV. Also, the left part shows the organization of the 3D-stacked memory slices (can be multi-layer) and its memory controllers. The right part of Fig. 3.3 shows a CMP having 64 cores $\{c_0, c_1, c_2, \cdots, c_{63}\}$ and these cores are connected using an 8×8 2D-mesh NOC.

Further, considering the CMP system model as shown in Fig. 3.3, the target 3D-stacked DRAM memory based CMP system can be modeled as core memory interconnection graph $CMIG(\mathbf{C_{cc}},\ \mathbf{E_{Ecc}})$. $CMIG(\mathbf{C_{cc}},\ \mathbf{E_{Ecc}})$ graph is a collection of N core vertices (or NOC-tile vertices or cores, interchangeably used throughout the thesis) $\mathbf{C_{cc}}(c_0,\ c_1,\ ...,\ c_{N-1})$ that are connected using the electrical edge set ($\mathbf{E_{Ecc}}$). Edges $e_{ecc}(c_i, c_j) \in \mathbf{E_{Ecc}}$ whenever there is an interconnection between core vertices c_i and c_j. The core vertex is formed by grouping the associated adjacent elements such as processing core, router, memory slice and its associated memory controller (if present and adjacent to the processor layer router).

Moreover, the right part of Figure 3.3, can be considered as an example of $CMIG$ graph corresponding to considered CMP system architecture. This graph is having 64 core vertices $\mathbf{C_{cc}}(c_0,\ c_1,...,\ c_{63})$ and their respective electrical interconnects. In the right part of Figure 3.3, all the plain shaded core vertices have a core, private caches (L1 and L2) and processor layer router. Similarly, all the dark shaded vertices have

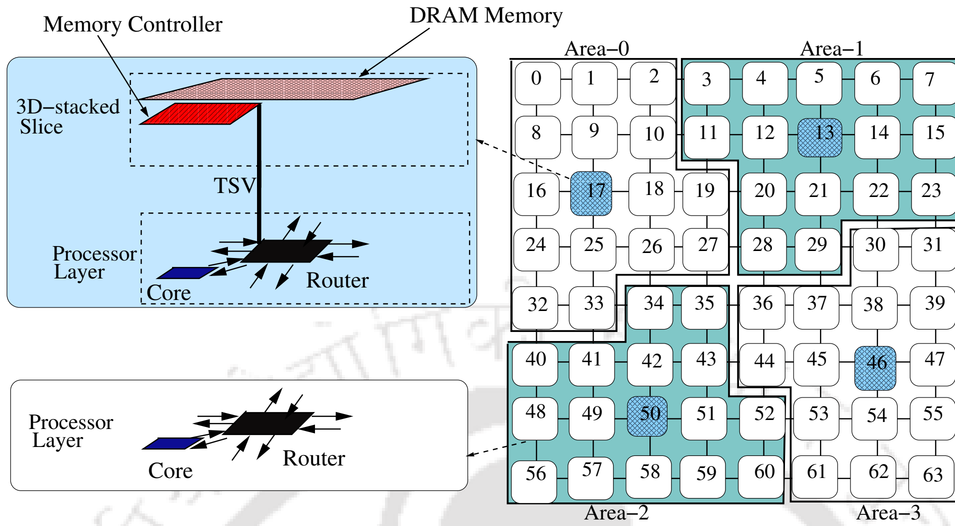


Figure 3.3: Example: Detailed system model representation for CMP with DRAM memory

a core, private caches (L1 and L2) along with the processor layer specific position router, and memory controller (along with its DRAM memory slice) together.

The above CMP system architecture having 3D-stacked DRAM memory and its core memory interconnection graph $CMIG$ representation is used in Chapter 4 and the first part of Chapter 5 to achieve our aforementioned objective.

3.1.2 DRAM and SRAM Buffer at the 3D-stacked Memory Layer

In this case, the CMP system has 3D-stacked DRAM memory along with the SRAM based buffer on top of the processor layer, which is similar to as explained in section 3.1.1 except the addition of the SRAM based buffer. Specifically, in this CMP system model, each DRAM memory slice has an SRAM based buffer (termed as mapping buffer, $Mbuff$) along with the memory controller (MC). Each DRAM memory slice along with the $Mbuff$ and MC is connected to a processor layer router (specific positioned router) using TSV. Each $Mbuff$ caches *the frequently accessed cache blocks* of the pages associated with its DRAM memory slice. The higher hit rate of $Mbuff$ (due to efficient interaction method with DRAM memory slice and its size) mitigates the high DRAM access latency.

TH-2297_136201001

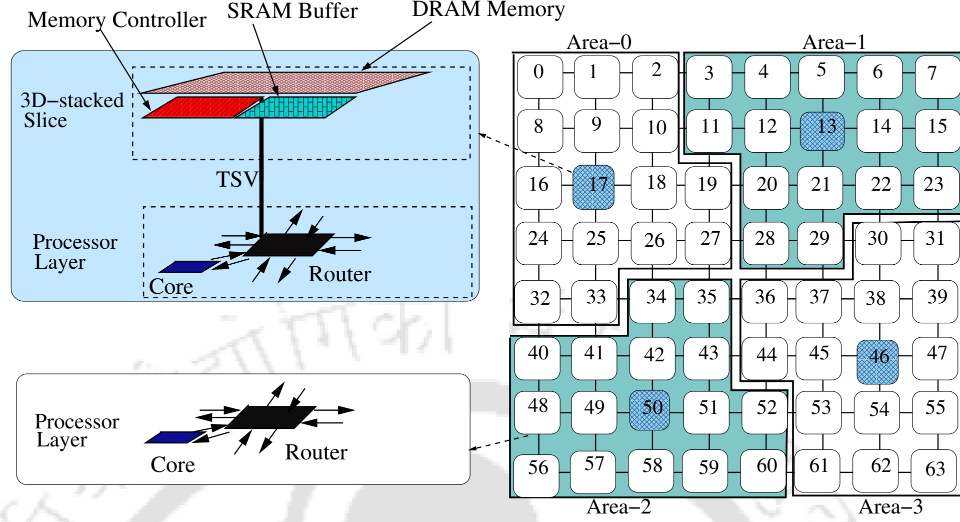


Figure 3.4: Example: Detailed system model representation for CMP with DRAM memory and SRAM buffer

Fig. 3.4 shows an example of the CMP system model having 3D-stacked DRAM memory along with the SRAM memory-based buffer as the constituent of the 3D-stacked memory slices. The right part of Fig. 3.4 shows a CMP having 64 cores $\{c_0, c_1, c_2, \cdots, c_{63}\}$ and these cores are connected using an 8×8 2D-mesh interconnect. Also, each core is associated with a router. Further, the left part (three-dimensional view) shows the 3D-stacked slice (associated to 3D-stacked layer) organization and its interconnection with the processor layer routers using TSVs. 3D-stacked layer on top of the processor layer is having four stacked DRAM slices m_0, m_1, m_2, and m_3. Each DRAM slice is having an on-chip MC and an SRAM memory-based buffer $Mbuff$, and these are connected to the specific routers placed at processor layer. This form of the CMP system model is used in the second part of Chapter 5 to perform a comparative study between self-adaptive run-time page mapping and a state-of-art work [30].

The CMP system model considered in this section is similar to the CMP system model as considered in section 3.1.1, except the additional $Mbuff$ placed at the 3D-stacked slice. Therefore, the $CMIG(\mathbf{C_{cc}}, \mathbf{E_{Ecc}})$ graph which is considered in section 3.1.1, can also be used for the target CMP system model of this section. So, considering the CMP system model, as shown in Fig. 3.4, the target 3D-stacked DRAM memory-based CMP system along with the SRAM buffer can be modeled as core

TH-2297_136201001

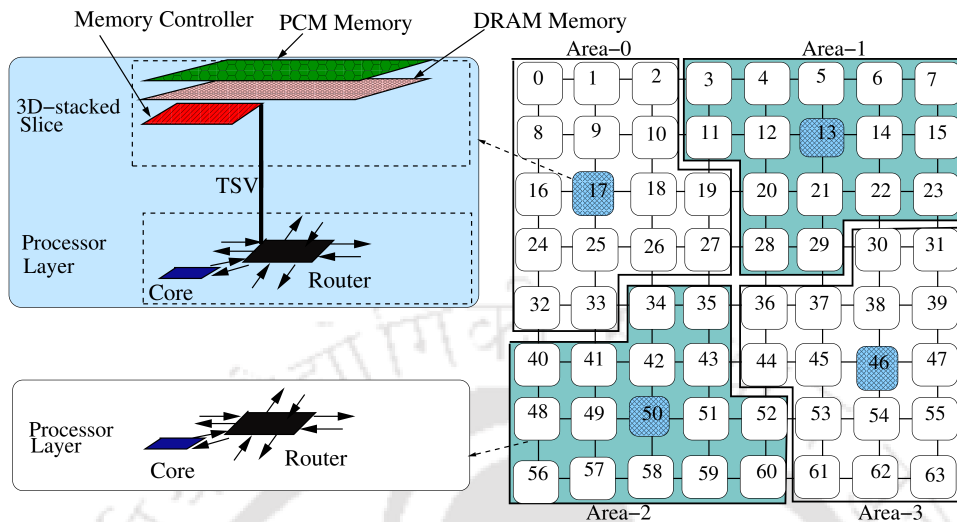


Figure 3.5: Example: Detailed system model representation for CMP with DRAM-PCM based hybrid memory

memory interconnection graph $CMIG(\mathbf{C_{cc}}, \mathbf{E_{Ecc}})$. Where, everything is same except the core vertex constituent elements, which also includes a $Mbuff$ along with the other elements.

3.1.3 DRAM and PCM memory at the 3D-stacked Memory Layer

In this case, the CMP system has 3D-stacked DRAM and PCM memory placed on top of the processor layer. 3D-stacked DRAM memory slice and PCM memory slice together form a hybrid memory slice, and together there are M number of hybrid memory slices. Also, each 3D-stacked hybrid memory slice is having an on-chip memory controller (MC) associated with it. This form of the CMP system model is used in Chapter 6 along with the details if required.

For example, the right part of Fig. 3.5 (two-dimensional view), shows a CMP having 64 cores $\{c_0, c_1, c_2, \cdots, c_{63}\}$ and these cores are connected using an 8×8 2D-mesh based interconnects. It shows the four specific positions denoted by c_{17}, c_{13}, c_{50}, and c_{46} associated to four 3D-stacked hybrid memory slices m_0, m_1, m_2, and m_3 respectively. The left part (three-dimensional view) of Fig. 3.5, shows the 3D-stacked multi-layer organization of the hybrid memory slice and its interconnection

TH-2297_136201001

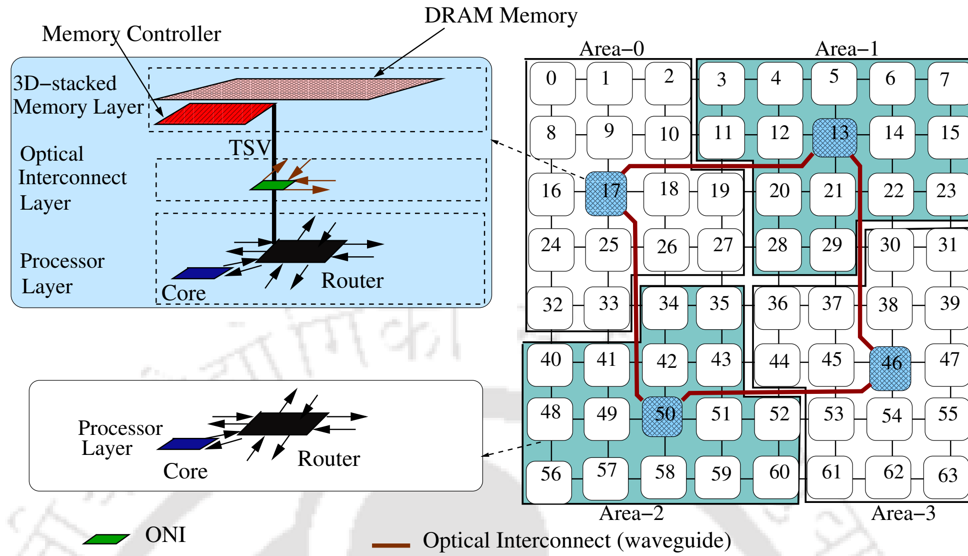


Figure 3.6: Example: Detailed system model representation for CMP with DRAM memory and optical interconnect

with the processor layer routers using TSV. The bottom layer of each memory slice has a memory controller associated with it. Also, each hybrid memory slice contains the DRAM memory as well as PCM memory and constitutes a hybrid memory slice.

3.1.4 DRAM Memory at the 3D-stacked Memory Layer along with an Optical Layer

In this form of the CMP system model, an **optical interconnect layer** in between the 3D-stacked memory layer and processor layer is placed. The optical interconnect layer is having M optical network interfaces (ONIs), and these are connected via optical interconnects (waveguides). Each ONI comprises micro-ring resonator and on-chip laser sources which are similar to as explained in Beux *et al.* [61]. Each ONI is connected to a processor layer router (specific position router in 2D-mesh) and a memory controller (associated with the DRAM memory slice) using the TSV. This form of CMP system model is used in Chapter 7.

Figure 3.6 shows an example and detailed representation of the considered 3D-stacked DRAM memory-based CMP having optical interconnect. In Figure 3.6, processor layer has 64 cores that are connected using an 8×8 two-dimensional mesh of electrical interconnects. Optical interconnect layer on top of the processor layer is

having four optical network interfaces (oni_0, oni_1, oni_2, oni_3 $\in$ **ONI**) and these are connected to each other through optical interconnects (waveguide), which is shown as thick line between the nodes in right part of Fig. 3.6. Further, a 3D-stacked DRAM memory layer is placed on top of the optical interconnect layer. There are four 3D-stacked DRAM memory slices (or memory banks) at the 3D-stacked memory layer $\{m_0$, m_1, m_2 and $m_3\}$ and each memory slice is having their own on-chip MC associated with it.

Researchers have reported that optical network interfaces (ONIs) and DRAM memory slices can be considered on-chip [67, 40]. Therefore, we can assume that communication time and distance from the memory slices as well as ONIs to processor layer routers using TSVs are negligible. So, the considered 3D-stacked DRAM memory based CMP having optical interconnect, can be modeled as the core memory hybrid interconnection graph $CMHIG(\mathbf{C_{cc}}$, $\mathbf{E_{Ecc}}$, $\mathbf{E_{Occ}})$. $CMHIG(\mathbf{C_{cc}}$, $\mathbf{E_{Ecc}}$, $\mathbf{E_{Occ}})$ graph is a collection of NOC-tile vertices $\mathbf{C_{cc}}(c_0$,c_1, ...,$c_{N-1})$ that are connected using an electrical edge set ($\mathbf{E_{Ecc}}$) and optical edge set ($\mathbf{E_{Occ}}$). The NOC-tile vertex is formed by grouping together the associated adjacent elements (processing core, memory controller and ONI, based on their presence in some cases) with each processor layer router. The left part of Figure 3.6 shows the constituent components of the NOC-tile vertices based on the associated adjacent elements to each processor layer routers. $e_{ecc}(c_i, c_j) \in \mathbf{E_{Ecc}}$ whenever there is an electrical interconnection between NOC-tile vertices c_i and c_j. Moreover, $e_{occ}(c_i, c_j) \in \mathbf{E_{Occ}}$ whenever there is an optical interconnection between NOC-tile vertices c_i and c_j.

Moreover, the right part of Figure 3.6, can be considered as an example of $CMHIG$ graph corresponding to considered CMP system architecture. This graph is having 64 NOC-tile vertices $\mathbf{C_{cc}}(c_0$, c_1,..., $c_{63})$ and their respective electrical and optical interconnects. In the right part of Figure 3.6, all the plain shaded NOC-tile vertices have a core, processor layer general-router (GR), and their private caches (L1 and L2) together. Moreover, all the dark shaded vertices have a core, private caches (L1 and L2), ONI, processor layer specific-router (SR) and, DRAM controller together. In Figure 3.6, left part shows an enlarge abstract view of the 55th and 46th vertices that are the examples of the plain-shaded and the dark-shaded vertices respectively.

TH-2297_136201001

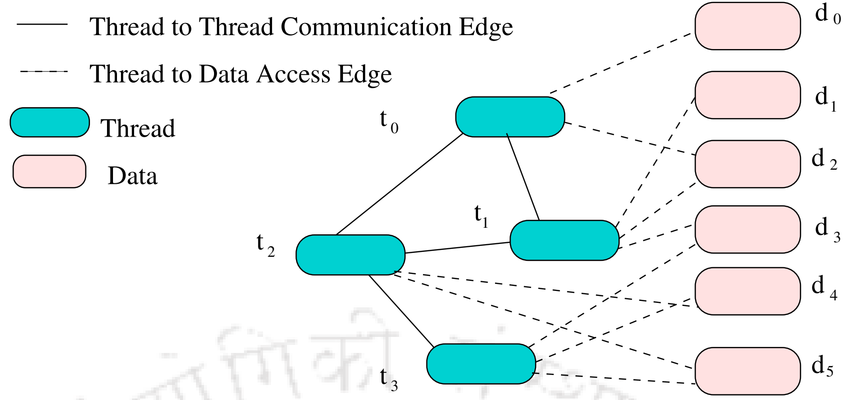


Figure 3.7: Example of application model

3.2 Application Model

It is evident that in multi-threaded applications, a single application is sub-divided into multiple threads based on their specific operations, and each thread can run in parallel to each other. Also, there exists data sharing in between the threads as they belongs to the same applications and uses the same variables. Therefore, considering the resource sharing properties of the multi-threaded applications between their independent threads, here we assume that a multi-threaded application is having N threads and D data, and N is same as the number of cores in the CMP. This assumption is valid if may run multi-threaded applications having multi-application workloads together to utilize all the processors of the CMP. A multi-threaded application can be represented as an application graph $AG(\mathbf{T}, \mathbf{D}, \mathbf{E_{tc}}, \mathbf{E_{da}})$. An application graph AG (corresponding to a multi-threaded application) consist of two types of nodes: (a) set of thread nodes $\mathbf{T}(t_0, t_1, ..., t_{N-1})$ and (b) set of data nodes $\mathbf{D}(d_0, d_1, ..., d_{D-1})$. These nodes are connected via two types of edges: (a) set of thread communication edges $e_{tc}(t_i, t_j) \in \mathbf{E_{tc}}$ and, (b) set of data access edges $e_{da}(t_i, d_j) \in \mathbf{E_{da}}$.

Fig. 3.7 shows an example of an application graph. This graph has four thread nodes (t_0, t_1, t_2, t_3), six data nodes (d_0, d_1, d_2, d_3, d_4 and, d_5) and their interconnection. The data access edge $e_{da}(t_i, d_j)$ represents the edge between the thread node t_i and data node d_j and the number of access to data d_j by the thread t_i is represented by the weight $\omega(e_{da}(t_i, d_j))$ of the edge. Similarly, thread to thread communication edges $e_{tc}(t_i, t_j)$ is between thread node t_i and t_j and weight $\omega(e_{tc}(t_i, t_j))$ represent the amount of data communicated between them.

TH-2297_136201001

In a virtual memory environment, a data node (or data) can be represented using either a virtual page or many virtual pages. For each virtual page vp_j, it get mapped to an associated physical page $pp \in \mathbf{PP}$ ($\mathbf{PP}$ is the set of all physical pages or frames) in the DRAM memory. So, the application graph $AG(\mathbf{T}, \mathbf{D}, \mathbf{E_{tc}}, \mathbf{E_{da}})$ can be converted to application graph with virtual paging $AGVP(\mathbf{T}, \mathbf{VP}, \mathbf{E_{tc}}, \mathbf{E_{vpa}})$. Where, term $\mathbf{VP}$ is the set of virtual pages (or total number of pages) corresponding to data of $\mathbf{D}$ and $\mathbf{E_{vpa}}$ is the edge set having access edges $e_{vpa}(t_i, vp_j) \in \mathbf{E_{vpa}}$ for the virtual page $vp_j \in \mathbf{VP}$ accessed by a thread t_i. Our assumption, the virtual page access by threads may include the virtual page accesses of whole virtual space of the application including data, code, heap and any other parts of the virtual memory.

Indian Institute of Technology Guwahati

TH-2297_136201001

4

Static Profile Based Mapping

In earlier days, SRAM technology was popular to implement caches inside the chip. However, recently, 3D die stacking technology became more advanced to fabricate on-chip high density 3D-stacked DRAM memory [69, 67, 40]. Research in [69], reported that placing memory on top of the processor using 3D fabrication technologies (3D-stacked memories) shows an impressive performance benefit of up to 92%. High capacity (in GBs) and bandwidth on-chip 3D-stacked DRAM memory has the potential to satisfy the growing need of the current as well as future generation CMP systems, where core count is high.

In this Chapter, a 3D-stacked DRAM memory-based CMP system is considered (target CMP system of this Chapter), where on-chip 3D-stacked DRAM memory is connected to the processor layer using TSV (to knock down the memory wall problem of the CMP system). Further, to avoid the coherence directory and remote memory access related issues for the large size 3D-stacked DRAM memory, we have proposed efficient profile based static application mapping on to the 3D-stacked DRAM memory based CMP. Specifically, in this Chapter, we have designed profile based static application mapping, where virtual pages of the application get mapped to the DRAM memory slices, and threads of the application get mapped to the cores of the CMP system such that on-chip communication can be minimized. This Chapter presents an alternative way to use the 3D-stacked DRAM memory for the CMP systems. In our proposed approach, 3D-stacked DRAM memory is considered as non-coherent memory. Moreover, to reduce the effective remote memory access related overheads, we perform a static virtual page to DRAM memory slice mapping.

TH-2297_136201001

4.1 Problem Formulation

Consider, the application graph representation $AGV(\mathbf{T}, \mathbf{VP}, \mathbf{E_{tc}}, \mathbf{E_{vpa}})$ and the target 3D-stacked DRAM memory based CMP representation $CMIG(\mathbf{C_{cc}}, \mathbf{E_{Ecc}})$, as given in Section 3.2 and 3.1.1 of Chapter 3 respectively. Along with the assumption that the number of threads of an application and the number of core in the CMP system is equal and it is N for both, as explained in Chapter 3.

Based on the target CMP system and multi-threaded application graphical representations, the aim of this Chapter is to find the mapping of N threads to the N cores and VP virtual pages to the M DRAM memory slices such that overall on-chip communication cost is minimized. In order to achieve this, two mapping functions or tables, X (thread to core table or TCT) : $\mathbf{T} \rightarrow \mathbf{C}$ and Y (virtual page to memory slice table or VMT) : $\mathbf{VP} \rightarrow \mathbf{M}$ need to be found such that sum of thread communication cost (C_{comm} represented by equation 4.1) and memory access cost (C_{mac} represented by equation 4.2) is minimized subjected to following:

- One thread is assigned to only one processing core.
- Many threads can access to one virtual page, and the virtual pages need to be in one of the DRAM memory slices.
- Many virtual pages can be mapped to one memory slice. In case if the system does not support virtual memory, then the size of all mapped pages should not exceed the size of the memory slice. However, nowadays, almost all the system supports virtual memory technique so that pages can be swapped in and out. Therefore, a memory slice can hold virtually infinite pages.

The thread communication cost C_{comm} is given as,

$$C_{comm} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \omega(e_{tc}(t_i, t_j)) \times distCC(X(t_i), X(t_j)). \quad (4.1)$$

Where $\omega(e_{tc}(t_i, t_j))$ is weight of the edge $e_{tc}(t_i, t_j)$ and this amount of data get communicated between thread t_i and t_j. Term $X(t_i)$ and $X(t_j)$ are the core vertices having threads t_i and t_j mapped to their cores respectively. The term $distCC(X(t_i), X(t_j))$ is the Manhattan distance between the core vertices having threads t_i and t_j mapped to their cores.

TH-2297_136201001

The memory access cost C_{mac} is given as,

$$C_{mac} = \sum_{i=0}^{N-1} \sum_{j=0}^{|\mathbf{VP}|-1} \omega(e_{vpa}(t_i, vp_j)) \times distCC(X(t_i), Y(vp_j)). \tag{4.2}$$

Where $\omega(e_{vpa}(t_i, vp_j))$ is the number of page access by thread t_i to page vp_j. $Y(vp_j)$ gives the $CMIG$ core vertex number associated to the DRAM memory slice having mapped page vp_j. The term $distCC(X(t_i), Y(vp_j))$ is the distance between $X(t_i)$ (the core vertex having thread t_i mapped to its core) and $Y(vp_j)$ (the core vertex associated to the memory slice having mapped page vp_j). Moreover, term **VP** represents the set of virtual pages or total number of virtual pages corresponding to the multi-threaded application. Terms $distCC(X(t_i), X(t_j))$ and $distCC(X(t_i), Y(vp_j))$ are calculated using the electrical edge set $\mathbf{E_{Ecc}}$ associated to the $CMIG$ graph of the target CMP system. In case of the considered $CMIG$ graph associated to the 3D-stacked DRAM memory based CMP (where cores are connected using 2D-mesh network), the value of $distCC(X(t_i), X(t_j))$ and $distCC(X(t_i), Y(vp_j))$ are hop-to-hop Manhattan distance between the associated core vertices. We have assumed 1 hop distance between any two adjacent core vertices throughout the thesis to calculate the distances $distCC(X(t_i), X(t_j))$ and $distCC(X(t_i), Y(vp_j))$.

As reported by Mutlu *et al.* in [38], NOC consumes about 40% of on-chip power. So, if we reduce the overall on-chip communication cost (C_{comm}+C_{mac}), it reduces the on-chip power consumption significantly, which makes it an essential motivational factor to solve this problem.

4.2 Static Profile Based Mapping

In this Section, we use profile driven mapping of the benchmarks to minimize the on-chip communication cost. The benchmarks considered are PARSEC [14], SPLASH-2 [118], and many standard multi-threaded applications written using Cilk [42]. In profile based mapping, the application is executed once on the target CMP to get the statistics about thread to thread communication, and thread to virtual page access count. The same profiled information is used to optimize the overall communication cost of the application for the next execution of the application, assuming that similar kind of data access pattern will be observed by a different run of the same application. This Section describes (a) simulated annealing based thread to core mapping, (b)

TH-2297_136201001

virtual pages to memory slices mapping, (c) thread mapping followed by virtual page mapping, and finally (d) combined thread and virtual page mapping. The first two mapping approaches (a) and (b) are to analyze the effects of the thread to core mapping and virtual page to memory mapping respectively. However, (c) and (d) are two different approaches that uses the thread to core and virtual page to memory mapping concepts together and their aim is to study the effects of the combination of both the mapping components.

4.2.1 Thread to Core Mapping

In thread to core mapping process, we map N threads of the multi-threaded application on to N cores of CMP such that core to core communication cost (given by equation 4.1) can be minimized. The mapping solution X need to be found, for which the C_{comm} is minimum. This mapping problem is exactly same as the quadratic assignment problem (QAP). The QAP problem is an NP-Hard problem, and there is no possibility of finding an approximation scheme for the same problem [60]. So meta-heuristics are used to solve this kind of problem, to get a good solution within a reasonable time. We have used simulated annealing (other heuristics also works, but simulated annealing is used because of simplicity and less execution time) meta-heuristics to solve this problem. Algorithm 1 shows pseudo-code for solving this problem using simulated annealing. Simulated annealing uses single solution based iterative refinement by single pair exchange neighbor generation and evaluation in solving the problem [60].

This algorithm starts with an initial solution (which is a random thread to core mapping table X, an example is shown in the top part of Table 4.1). Every iteration of the algorithm generates a neighbor solution (thread to core mapping table X) using the exchange of a randomly selected pair ($X[a]$ and $X[b]$), and evaluates new cost C^{new}_{comm} using Equation 4.1. The algorithm accepts a newly generated solution if the cost C^{new}_{comm} associated with the new solution is less than the cost C_{comm} associated with the current best solution. The new solution does get accepted even if the cost of the new solution is higher than the cost of the current best with some probability defined by $\frac{random()\%C^{new}_{comm}}{C^{new}_{comm}} \leq e^{-\frac{C^{new}_{comm}-C_{comm}}{Temp}}$ (part of step-7). Therefore, the new solution X with increased cost also get accepted with some probability. The probability of acceptance of solution with higher current cost depends on temperature value at the current iteration, and it decreases with iteration. In each iteration temperature value

TH-2297_136201001

Algorithm 1 :Thread to Core Mapping Using Simulated Annealing

1: Initialize: X (Thread to Core mapping table), and Temp=HIGH;
2: C_{comm} = Evaluate cost using Equation 4.1 and X.
3: **while** $Temp > LOW$ **do**
4: a=random()%N, b=random()%N
5: Generate a neighbor solution value of X by $Swap(X[a], X[b])$
6: C_{comm}^{new} = Evaluated cost using new X and Equation 4.1
7: **if** $(C_{comm}^{new} < C_{comm}) \mid\mid (\frac{random()\%C_{comm}^{new}}{C_{comm}^{new}} \leq e^{-\frac{C_{comm}^{new} - C_{comm}}{Temp}})$ **then**
8: $C_{comm} = C_{comm}^{new}$
9: **else**
10: $SwapBack(X[a], X[b])$
11: **end if**
12: Reduce temperature : Temp $=\alpha \times$ Temp;
13: **end while**
14: return X

Initial thread to core mapping															
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
c_{15}	c_{14}	c_{10}	c_3	c_4	c_8	c_6	c_7	c_5	c_9	c_2	c_{11}	c_{12}	c_{13}	c_1	c_0
Optimized thread to core mapping using SA (simulated annealing)															
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
c_{13}	c_4	c_{10}	c_5	c_8	c_{15}	c_3	c_7	c_6	c_0	c_9	c_{14}	c_{11}	c_2	c_{12}	c_1

Table 4.1: Initial and optimized thread to core mapping

reduce by a factor of α, and the typical value of $\alpha = 0.9$. Value of lowest temperature ($LOW = 1$), highest temperature ($HIGH = 1000$) and α controls the number of iterations. Simulated annealing produces a reasonably good solution $X[N]$ (mapping of thread to the core) in less time. The bottom part of Table 4.1 shows an example of the result produced by this method.

4.2.2 Mapping of Virtual Pages to DRAM Memory Slices

In memory page mapping, the on-chip communication cost of virtual page access from all the cores is minimized as described in Equation 4.2. The values of the number of virtual pages ($|VP|$), the number of cores or threads (N) and the number of DRAM memory slices (M) are not same. Typically these values have relation $|VP| >> N > M$ as in contrast to the earlier case of thread to core mapping where the number of cores and the number of threads are equal.

TH-2297_136201001

Algorithm 2 :Virtual Pages to DRAM Slice Mapping

1: Initialize: Y (Virtual page to DRAM slice mapping table), X (Thread to Core mapping table).
2: **for** every virtual page vp_j ($j \leq |VP|$) **do**
3:    **if** $\omega(e_{vpa}(t_i, vp_j))$ is not zero for all the threads $t_i \in N$ **then**
4:       $Cost = \infty$.
5:       **for** All the memory slices $m_j \epsilon M$ **do**
6:          Y'=$Y(vp_j)$.
7:          Map virtual page vp_j to m_j i.e. $Y(vp_j)$=m_j.
8:          $CostNew$=$\sum_{i=0}^{N-1} \omega(e_{vpa}(t_i, vp_j)) \times distCC(X(t_i), m_j)$.
9:          **if** $CostNew < Cost$ **then**
10:             $Cost = CostNew$.
11:          **else**
12:             $Y(vp_j)$=Y'.
13:          **end if**
14:       **end for**
15:    **end if**
16: **end for**
17: return Y.

So, with these above simplifications, the virtual page mapping problem can be solved using a simple approach. Algorithm 2 shows the pseudo-code for mapping of virtual pages to DRAM memory slices. This method takes active virtual pages (active virtual page means $\omega(e_{vpa}(t_i, vp_i))$ is not zero for all the threads t_i) one by one and map to a DRAM memory slice which minimizes the on-chip memory access cost of that page and so minimizes the memory access cost defined by Equation 4.2.

In the evaluation of memory mapping, we use the default thread to core mapping (X) that is a random mapping of thread to the core as shown in the upper part of the Table 4.1. As $|VP| >> N > M$, the on-chip communication due to memory access has a significant share as compared to the core to core communication in the overall on-chip communication cost (summation of C_{comm} and C_{mac}). Hence efficient virtual page mapping to DRAM memory slice reduces the overall on-chip communication cost significantly.

As an example, Figure 4.1 shows the percentage of active virtual pages mapped to different memory slices with and without page mapping to DRAM slices for some randomly selected benchmark applications on 4×4 CMP system having 4 DRAM slices. The change in the percentage of virtual pages mapped to different DRAM memory slices is significant before and after the virtual page mapping optimization.

TH-2297_136201001

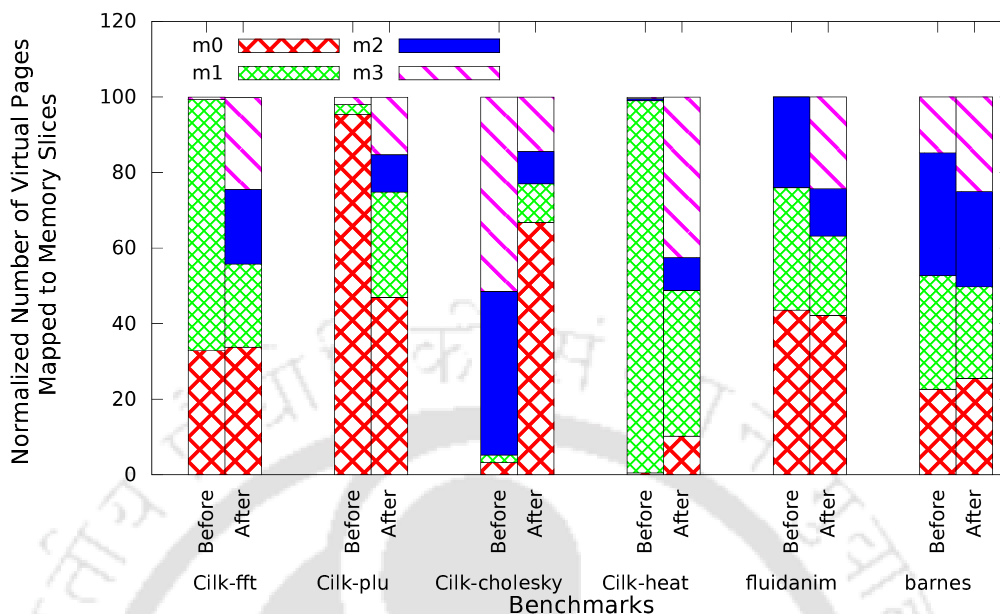


Figure 4.1: Percentage of active virtual pages mapped to different memory slices: before and after page mapping

For Cilk-fft benchmark the most of the virtual page get mapped to DRAM slice m_0 and m_1, but after virtual page mapping optimization, the pages are getting distributed to all the DRAM slices. Similarly, for other benchmarks, the virtual pages get mapped to different DRAM slices to minimize the overall on-chip communication cost due to memory access cost.

4.2.3 Thread Mapping Followed by Virtual Page Mapping

In this case, we generate the optimal thread to core mapping using Algorithm 1, and after that virtual page to DRAM slice get mapped by the approach described in the previous Sub-section or Algorithm 2. In this case, thread to core mapping gets higher priority as compared to the virtual page to DRAM slice mapping.

4.2.4 Combined Thread Mapping and Virtual Page Mapping

Communication cost due to the virtual page accessed by the threads dominates the overall on-chip communication cost, and also many threads access a virtual page. So the mapping of both threads to cores and virtual pages to DRAM slices together may be crucial. In this case, we map (thread, virtual page) access pairs to be in adjacent cores and DRAM slices together.

TH-2297_136201001

In this mapping process, we create a list of communications for the thread to thread pairs and thread to virtual page accesses pairs. Communication pair consists of a source, destination, and amount of data.

- For thread to thread pairs, the communications are in the form (t_i, t_j, $e_{tc}(t_i,t_j)$), where $e_{tc}(t_i, t_j)$ is the data communication edges. We list out this form of communications for all the thread pairs.
- For thread to virtual page access pairs, the communications are in the form $(t_i, vp_j,\ e_{vpa}(t_i, vp_j))$, where $e_{vpa}(t_i, vp_j)$ the virtual page access edges. We list out this form of communications for all the threads to all the virtual page access pairs.

We sort all the thread-thread communication pairs (based on $\omega(e_{tc}(t_i, t_j))$) and all the thread to virtual page access pairs (based on $\omega(e_{vpa}(t_i, vp_j))$) together in non-increasing order of their amount. The combined mapping approach uses this sorted communication pairs. Combined thread to the core and virtual page to DRAM slice mapping employs the following three phases:

(a) Phase I: In this phase of mapping, we select some top $(t_i, vp_j,\ e_{vpa}(t_i, vp_j))$ communication pairs from the sorted combined list of communication pairs and map to the resources till all the DRAM slices get at least one mapped virtual page. Initially, we chose M pairs of thread to virtual page access $e_{vpa}(t_i, vp_j)$ and map their vp_j to different DRAM slices and their thread to core which is near to the DRAM slices (or core which share the router with DRAM slices). For an example, considering the CMP system model as shown by Figure 3.3, we map top four thread-virtual page pairs to (c_{17}, m_0), (c_{13}, m_1), (c_{50}, m_2) and (c_{46}, m_3) such that the overall communication cost is minimum.

    If among the top M pairs have common threads then we map both the virtual pages to one DRAM slice and the common threads get mapped to nearby one and near to the allocated memory slices. Similarly, if the top M pairs have a common virtual page, then we assign the common virtual pages to one DRAM slice and the threads to nearby one and near to the memory slices. If this happens, then include more $e_{vpa}(t_i, vp_j)$ pair to cover all the M DRAM slices.

(b) Phase II: In this phase of mapping, we select communication pairs from the sorted, combined list of communication pairs one by one and map to the resources till all the threads assigned to one core. Till all the threads not mapped

TH-2297_136201001

to core, we select a top pairs from sorted combined list of communication pairs which is either $e_{tc}(t_i, t_j)$ or $e_{vpa}(t_i, vp_j)$ communication pair and try to map them. One of the item (thread or virtual page) of the pair is already mapped then map the other item of the pair accordingly.

If the next selected pair is $e_{tc}(t_i, t_j)$ pair then we allocate them nearby cores for thread t_i and t_j. And similarly if the next selected pair is e_{vpa} (t_i, vp_j) pair then we find a core-DRAM slice pair where distance is minimum and allocate them.

(c) Phase III: In the last phase, we use simple virtual page to DRAM slice mapping as described in Section 4.2.2 and Algorithm 2 for rest of the unmapped $e_{vpa}(t_i, vp_j)$ pairs.

Algorithm 3 shows the pseudo-code for combined mapping of the virtual page to DRAM slices and thread to cores. If more than one core is at the same distance from the MC during thread mapping, then randomly any one core is considered. The first phase, second phase, and the last phase of mapping get performed in line number 4 to 11, 12 to 18 and 19 of Algorithm 3 respectively.

4.3 Experimental Setup

We have used Sniper simulator (version 6.1 [23]) to evaluate our proposed mapping methodologies for mapping multi-threaded benchmarks onto the 3D-stacked DRAM based CMP. Sniper simulator uses Pin binary instrumentation tool and can simulate X86 binary of multi-threaded benchmark applications. We have evaluated our proposed mapping approaches by using many PARSEC benchmarks [14], SPLASH-2 [118] benchmarks, and many other standard multi-threaded application written using Cilk [42]. We run these benchmarks on the simulator by using 16-cores (4×4 two-dimensional mesh and 4 MCs), 36-cores (6×6 two-dimensional mesh and 5 MCs), and 64-cores (8×8 two-dimensional mesh and 8 MCs) multiprocessor configurations to gather the profile data. Table 4.2 shows the micro-architecture parameters used in this paper. The profile data of a benchmark contains core to core communication traces (the amount of data communication) and core to L3 cache miss traces (which is, in general, get converted into the thread to virtual page access frequencies).

In modern-day systems, operating system (OS) uses address space layout randomization (ASLR) to improve memory protection process to safeguards against

TH-2297_136201001

Algorithm 3 :Combined Virtual Page and Thread Mapping

1: Initialize: X (Thread to Core mapping table) and Y (Virtual page to DRAM slice mapping table).
2: Create a list of communications including both thread to thread communication $(t_i, t_j, e_{tc}(t_i, t_j))$ and thread to virtual page access $(t_i, vp_j, e_{vpa}\ (t_i, vp_j))$ communication.
3: Sort the both list of communication pairs together in non-increasing order of their weight (or cost).
4: //******** Phase 1 ********//
5: **while** All the DRAM slices not got mapped at least one virtual page **do**
6: Take the top $(t_i, vp_j, e_{vpa}\ (t_i, vp_j)$ pair.
7: **if** The chosen pair have thread value already mapped **then**
8: Map the virtual page to the DRAM slice near to the mapped cores.
9: **else if** The selected pair has a virtual page already mapped **then**
10: Map the thread to the nearest free core to the DRAM slice of the virtual page.
11: **else**
12: Choose a un-mapped DRAM slice, map the virtual page to the DRAM slice and thread to core near to the DRAM slice.
13: **end if**
14: **end while**
15: //******** Phase 2 ********//
16: **while** All the thread not got mapped **do**
17: Take a top pair from sorted combined list of communication pairs.
18: **if** One of item (thread or virtual page) of the pair is already mapped **then**
19: Map the other item of the pair accordingly.
20: **else**
21: If the pair is a $e_{vpa}(t_i, vp_j)$ pair then find a core-DRAM slice pair where distance is minimum and allocate them.
22: If the pair is a $e_{tc}(t_i, t_j)$ pair then allocate them to nearby cores.
23: **end if**
24: **end while**
25: //******** Phase 3 ********//
26: Map rest of the un-mapped virtual pages using Algorithm 2.
27: return X and Y

buffer overflow attacks. ASLR randomizes the location where system executables are loaded into memory. In the ASLR process, the OS map different virtual address to data for various executions of the same executable by just adding a random number of pads to stack and heap area. In such cases, $E_{vpa}(t_i, vp_j)$ will be different for different execution of the same application. So in the profile based application mapping, we disable ASLR to ensure that for every run of the application, application's data

TH-2297_136201001

get mapped to the same virtual addresses.

In general, the simulated processor of the Sniper simulator generates the 64-bit virtual address. This 64-bit virtual address gets converted to newer form of virtual address, with 16-bit *application id*, 36-bit *virtual page number* and 12-bit *offset* for 4 KB page size (as mentioned and used in Sniper simulator [23]). The higher 24 bits of 36-bit virtual page number get randomized using an address randomization table which contains each 0 to 255 number only once in the same order. From this randomized address, 32nd and 33rd bit decides the DRAM slices and this mapping is static (the values 00, 01, 10 and 11 for DRAM slice number zero, one, two, and three respectively for the system with 4 DRAM slices). So the fixed part of the virtual address is lower 24-bits, which means a chunk of 16 MB data (2^{12} continuous virtual pages) get mapped to a DRAM slice. However, in our case, we modify the simulator to use a 12-bit page offset, 20-bit virtual page number and 32-bit tag bit, and a virtual page can go to any DRAM slices by our mapping process.

To evaluate our profile-based mapping, we require to include the thread to core mapping table (TCT: thread to core table, or X as described in Section 4.1), and virtual page to the physical page table (VMT: virtual page to memory slice table or Y as outlined in Section 4.1). The inclusion of the thread to the core mapping table (TCT) for an application is done at the application level at the time of the application execution. We have modified the source code of PARSEC, SPLASH-2 and, Cilk runtime scheduler to take the thread-to-core-mapping table (TCT) as one of the optional inputs, and it uses that. However, for providing user-defined page table (or the VMT, virtual page to DRAM slice) for the application, we had modified the simulator to accept the VMT table at the time of the simulation. The modified simulator uses our user-supplied page table for address translation and page mapping so that the virtual page get mapped to the desired DRAM slice of considered configuration.

4.4 Result and Overhead Analysis

4.4.1 Result Analysis

Figure 4.2, 4.3, and 4.4 show normalized on-chip communication cost for the different thread to core mapping and virtual page to physical page techniques for benchmarks on 4×4, 6×6, and 8×8 mesh architecture respectively.

TH-2297_136201001

Parameters	Values
NOC Topology	2D Mesh
Processor IPC	1
NOC link bandwidth	16 (bits/cycle)
Size of Processor load-store Queue	8
Overall on-chip 3D-stacked DRAM memory size	1 GB
Cache replacement policy (L1, L2, L3)	LRU
Cache size (L1, L2, L3) in KB	(32, 64, 512 respectively)
DRAM directory type	Full map
Cache coherence protocol	MOESI
Memory Contention considered	No

Table 4.2: Default CMP system configuration parameters

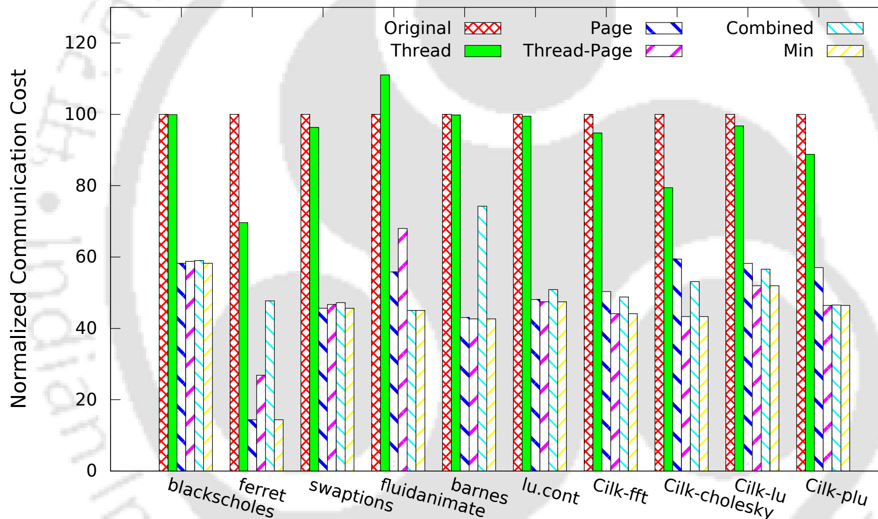


Figure 4.2: Normalized communication cost on 4 $\times$ 4 CMP system

The "Min" values in the bar graphs shown by Figures 4.2, 4.3, 4.4, and 4.5 represents the normalized overall communication cost having maximum reduction due to the applied mapping techniques that suit the benchmarks in the particular figure. "Original" is used to represent the normalized overall communication cost when there is no optimization is used, in this case, thread to the core and virtual page to memory slice mapping used are system generated. Moreover, "Thread," "Page," "Thread-Page," and "Combined" are used to represent reduced normalized overall communication cost due to the thread to core mapping, virtual page to memory slice mapping, virtual page mapping followed by thread mapping and, combined thread and virtual page mapping respectively.

TH-2297_136201001

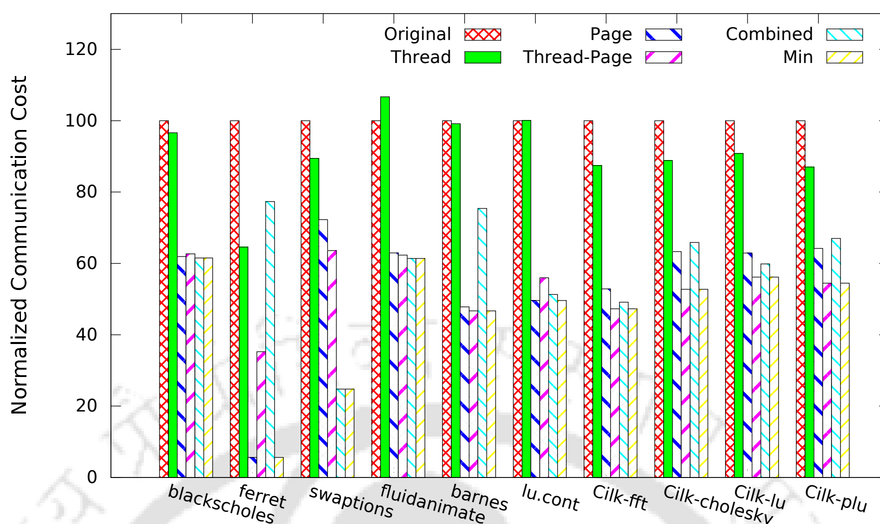


Figure 4.3: Normalized communication cost on 6 × 6 CMP system

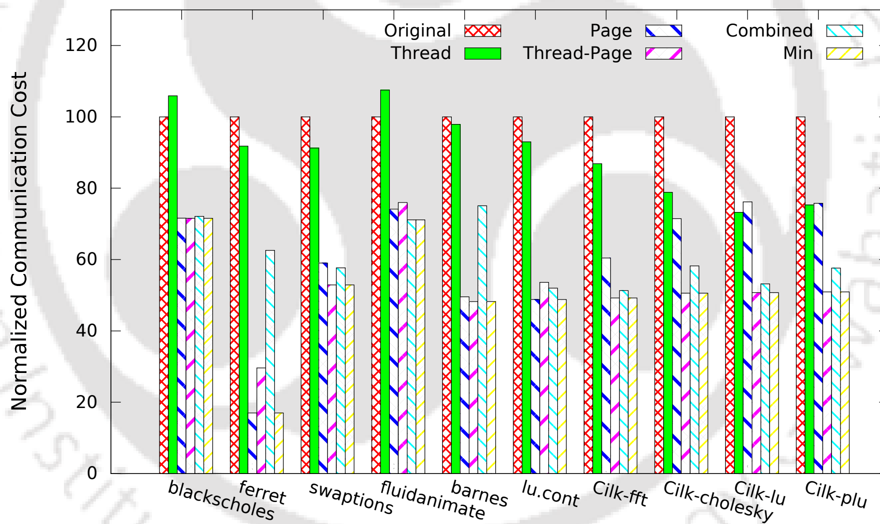


Figure 4.4: Normalized communication cost on 8 × 8 CMP system

As the memory access cost dominates the overall communication cost, only thread to core mapping is not effective. Also, for some cases, the overall communication cost increases due to only thread to core mapping, and this can be seen from Figure 4.2. The overall communication cost for "fluidanimate" benchmark increase when we apply the thread to core optimization as compared to default thread to core mapping. As the "Thread" only mapping considers number of communications between thread to thread and completely ignores the communication due to memory page accesses by the threads. Therefore, for some benchmarks, "Thread" only

TH-2297_136201001

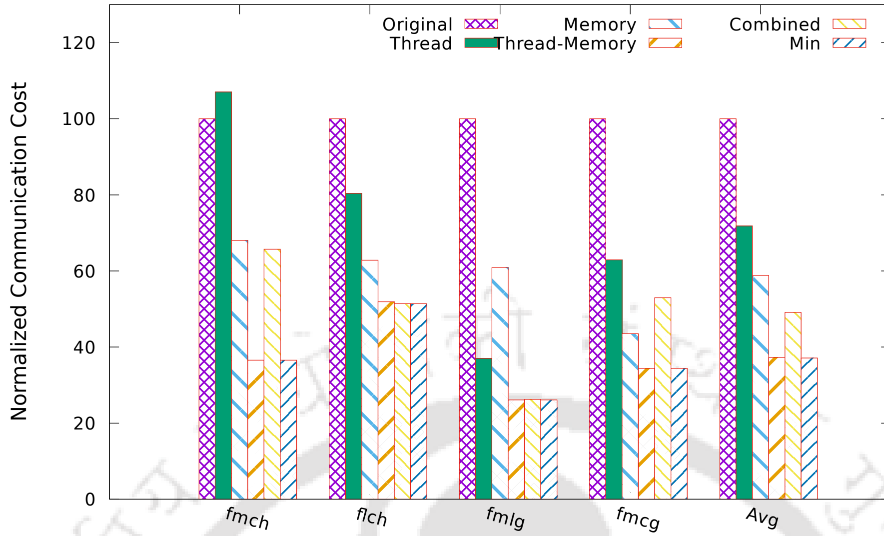


Figure 4.5: Normalized communication cost for multi-application workload, where f, c, h, l, g and m are fft, cholesky, heat, lu, magic and matmul benchmark respectively from Cilk

mapping reduces the thread communication cost by minimizing the distance between threads. However, hop-to-hop distance between threads and memory pages increases due to the thread only mapping, which results in the increase of the page access communication cost. Therefore, for some benchmarks, overall communication cost (which is sum of thread communication cost and page access communication cost) increases.

When we use the virtual page to DRAM slice mapping optimization, almost for all the benchmarks, the overall communication cost reduces the minimum of 26% and the maximum of 86% with an average of 56% for all 16, 36 and 64 core CMP configurations.

We have also tested our approach on mixed workload (randomly chosen from Cilk benchmarks) on 4×4 system to check the benefits due to virtual page mapping, and Figure 4.5 show a significant improvement. For mix multi-threaded workloads, the overall communication cost reduces a minimum of 47% and a maximum of 74% with an average of 64% for 16 cores configuration.

Thread to core mapping (using the simulated annealing method) reduces the overall on-chip communication cost up to 26% and an average of 12%. Whereas, other better thread to core mapping techniques (for example using Ant colony optimization (ACO) [41]) can give only 15% to 20% better result (with large execution

TH-2297_136201001

time overhead) as compared to this approach. As a virtual page to DRAM slice mapping dominates the overall communication cost, any good approach of thread to core mapping solve our purpose.

Fig. 4.6, shows that overall execution time of the applications before and after the virtual page mapping on 8×8 CMP system. Here, reduction in the overall application execution time is almost negligible as compared to the on-chip communication cost, as the DRAM memory access latency is dominating the hop to hop traversal latency.

The overall on-chip communication cost does not include any dominating parameter as overall execution time is having. The overall execution time parameter includes three main components, (1) time elapsed at processing core (includes private cache access time, time to process the data etc. at the processor), (2) time elapsed during on-chip network traversal, and (3) time elapsed during DRAM memory access.

Moreover, the virtual page mapping only reduces the on-chip hop-to-hop network traversal by reducing the remote accesses, and not the DRAM memory access time as well as execution time at the processor. So, when we consider the overall execution time optimization then the DRAM memory access time along with the execution time at the processor dominates the on-chip traversal time. Therefore, the effect of virtual page mapping on overall execution time is not that much significant as compared to the on-chip communication cost. If the hop-to-hop network traversal time dominates the DRAM access time as well as execution time at the processing core then the improvement in execution time is going to be significant. In the next chapter, we further explore this performance parameter by using the SRAM buffer.

As reported in [38], NOC consumes 40% power of large CMP, so the reduction of overall on-chip communication cost by an average of 56% and 64% reduces power consumption by an average of 22% and 27% for single applications and multi-application respectively.

4.4.2 Overhead Analysis

In the profile based mapping, we require two profiling counters to get the core to core communication amount E_{tc} and core to virtual page access count E_{vpa}. Virtual page to physical page mapping algorithm is also need to be changed a bit. In our case, as we have assumed M DRAM slices in the system (total DRAM of 1GB), so the size of each DRAM slice is $\lceil 1/M \rceil$ GB. The operating system page replacement algorithm needs to allocate a frame for the required logical page from the specified DRAM slices.

TH-2297_136201001

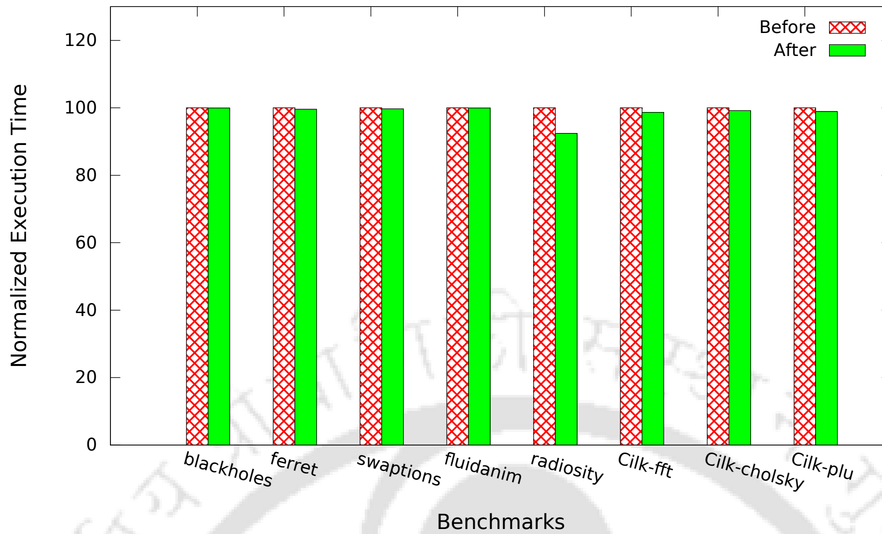


Figure 4.6: Normalized overall execution time on 8 × 8 CMP system

In table Y (result of virtual page to DRAM slice mapping), for the virtual pages $vp \in$ **VP** has any one of the corresponding value from $\{0, 1, 2, \ldots, (M-1)\}$ which specifies the DRAM slices to which vp is mapped.

4.5 Summary

In this chapter, different types of thread to the core and virtual page to DRAM slice mapping have been performed. The evaluation result shows that virtual page to DRAM slice mapping is more effective as compared to the thread to core mapping in overall communication cost reduction. Virtual page to DRAM slice mapping and thread to core mapping reduces overall on-chip communication cost up to 86% (average 56%) and 26% (average 12%) respectively. Moreover, in the next chapter, we perform the self-adaptive run-time application mapping and consider communication cost along with the execution time as the performance metric, and try to minimize them.

5

Self-adaptive Run-time Page Mapping

Clearly, there are two approaches to make the computation faster in the multiprocessor environment, one is to schedule the thread on to the core where data is available and other is migrate the data where thread is requesting. There is a trade-off between these approaches based on the target application and multiprocessor system. In the large cluster systems with many parallel threads and running with huge amount of data, balancing mix type threads (i.e compute bound, memory bound and i/o bound threads) between the processors is beneficial [120]. Therefore, for such systems, run-time thread migration to perform the balancing of the threads gives higher performance.

However, in our case, the target applications are shared multi-threaded in nature and need to be run on large CMP systems having larger 3D-stacked memories. Therefore, run-time thread mapping does not improves the system performance very significantly. The conclusion of Chapter 4 explains the same, which makes it clear that the overall communication cost reduction due to the virtual page mapping is significantly higher as compared to the reduction due to the thread mapping.

In addition, authors in [47], have described that the overheads associated with destroying a thread, transferring thread state (consist of a program counter, a set of registers, and a stack of procedure records containing variables local to each procedure), creating a new thread and initiating remote execution make run-time thread migration relatively tricky. Also, in phase-wise dynamic run-time mapping the num-

TH-2297_136201001

ber of phases can be large and performing thread to core mapping those many times may cumulatively cause larger overheads. Therefore, in this chapter, we consider only virtual page to memory mapping along with the default system generated thread to core mapping.

Research works in [29, 11, 105], shows that most of the application exhibits phase-wise behavior during their run-time. In other words, the application run-time execution manifests similar behavior within each phase and shows distinct characteristics between different execution phases. Therefore, the profile-based static mapping may not always be suitable for the complete duration of the application execution. Moreover, in the profile based static mapping, the application needs to be run at least once to get the profiled data.

Therefore, based on the facts (a) overall communication cost (C_{comm}+C_{mac}) is dominated by C_{mac}, (b) costly thread migration, (c) phase-wise behavior, and (d) unsuitable static mapping, in this Chapter we consider a 3D-stacked DRAM based CMP system and propose a self-adaptive run-time page mapping technique to mitigate the larger network traversal latency incurred while accessing the remote data. Further, to reduce the larger DRAM access latency, we have also proposed the use of an auxiliary small SRAM buffer and squarely performed a comparative study with the coherent DRAM cache model [30]. The auxiliary SRAM buffer ($Mbuff$, and termed as mapping buffer) stores the frequently accessed cache blocks and works as a cache of the 3D-stacked DRAM memory slices. Therefore, the auxiliary SRAM buffer causes the self-adaptive run-time page (data) mapping to be more effective by turning a DRAM access latency into SRAM access latency.

5.1 Problem Formulation

Consider, the multi-threaded application representation model $AGVP(\mathbf{T}, \mathbf{VP}, \mathbf{E_{tc}}, \mathbf{E_{vpa}})$, as given in Section 3.2 of Chapter 3. In addition with the target CMP representation model $CMIG(\mathbf{C_{cc}}, \mathbf{E_{Ecc}})$ associated to the (a) 3D-stacked DRAM based CMP and (b) 3D-stacked DRAM based CMP having SRAM buffer, as given in Section 3.1.1 and 3.1.2 of Chapter 3 respectively. The aim of our run-time mapping approach is to map the application $AGVP$ on to target architecture $CMIG$ to improve the performance. As communication cost C_{mac} due to the page mapping dominates the communication cost C_{comm}, so we need to design a run-time page mapping tech-

TH-2297_136201001

nique that considers the amount of thread to virtual page access edges $e_{vpa}(t_i, vp_j)$ $\in \mathbf{E_{vpa}}$ and ignores the thread to thread communication edges $e_{tc}(t_i, t_j) \in \mathbf{E_{tc}}$ of the $AGVP(\mathbf{T}, \mathbf{VP}, \mathbf{E_{tc}}, \mathbf{E_{vpa}})$. In run-time page mapping, a page may get mapped to different memory slices at different time instances of the application execution. So, value of $Y[vp_j]$ (defined in last Chapter 4) may not be fixed during the application execution.

Given page access requests (for cache blocks) to page vp_j by thread t_i at a time instant t, we formulate (1) total time $T_{L2M,total}$, and (2) total communication cost C_{Cost} due to the memory page accesses generated after the last level cache misses over the application run-time. Our objective in this chapter is to reduce the total time T_{Total} and total communication cost C_{Cost} due to the memory page accesses after the run-time page mapping.

The miss latency $T_{miss}(t_i, vp_j)$ incurred for thread t_i due to the last level cache (LLC) miss for the page vp_j at time instant t, can be calculated as follows.

$$T_{miss}(t_i, vp_j) = distCC(X(t_i), Y_t(vp_j)).L_{H2H} + [L_{MB} + MR_{MB}.L_{DRAM}]. \tag{5.1}$$

Where $X(t_i)$ is the core vertex associated to the core having mapped thread t_i. $Y_t(vp_j)$ is the core vertex adjacent to DRAM memory slice having mapped page vp_j at a time instant t. Term $distCC(X(t_i), Y_t(vp_j))$ is the Manhattan distance between core vertex associated to $X(t_i)$ and $Y_t(vp_j)$. The term L_{MB} is the $Mbuff$ access latency to access the block from the $Mbuff$, and the term L_{DRAM} is the DRAM access latency to access the cache block of a page. MR_{MB} is the $Mbuff$ miss rate and L_{H2H} is the hop to hop traversal latency. The latency to access a cache block means the time to get the cache block from higher level memory to the requested place.

Further, as most of the applications exhibit run-time phase-wise behavior, for simplicity suppose that the run-time of an application can be divided into multiple phases (or epochs) of fixed length. Therefore, using Equation 5.1, the total latency overhead $T_{epo,i}$ in a time epoch *epo* incurred (due to inter-node latency and memory read/write latency) while serving the L2 cache misses from thread t_i (mapped to core

TH-2297_136201001

c_i) can be given as follows

$$T_{epo,i} = \sum_{j=0}^{|\mathbf{VP}|-1} \omega_{epo}(e_{vpa}(t_i, vp_j)).\Big[T_{miss}(t_i, vp_j)\Big], \tag{5.2}$$

Where $\omega_{epo}(e_{vpa}(t_i, vp_j))$ is the number of page access request generated due to the L2 cache misses from thread t_i to the virtual page vp_j in a time epoch *epo*. Term **VP** represents the set of virtual pages or total number of pages corresponding to the multi-threaded application.

For an application with E number of phases (or time epochs) of the total execution time, the overall sum of miss latencies $T_{OverAll,i}$ associated with the L2 cache miss from the thread t_i (mapped to core c_i) is given as follows

$$T_{OverAll,i} = \sum_{epo=1}^{E} T_{epo,i}. \tag{5.3}$$

Therefore, considering the parallel execution of the multi-threaded applications on to the CMP and overlapped long latency L2 cache misses from each core $t_i \in T$, the total time (due to the memory accesses) $T_{L2M,total}$ of the application can be given as following:

$$T_{L2M,total} = Max(T_{OverAll,i}, \forall i \in \{0, 1, 2, \ldots, N-1\}). \tag{5.4}$$

Where Max gives the maximum value among all the $T_{OverAll,i}$ values. Therefore, $T_{L2M,total}$ is a reasonable estimate even if many page request happens in parallel from different threads. The goal of the mapping and $Mbuff$ is to reduce the $T_{L2M,total}$ of the CMP system.

The research in [51] revealed that the NOC consumes about 28% of the on-chip power considering the Intel Terascale 80-core chip multiprocessor. Also, the result of Chapter 4 shows that the communication cost due to the memory page access dominates the on-chip communication cost associated with the thread to thread communication. Therefore, the communication cost due to the memory page access can be evaluated by (5.5).

TH-2297_136201001

$$C_{Cost} = \sum_{epo=1}^{E} \sum_{i=0}^{N-1} \sum_{\forall j \in \mathbf{VP}} \omega_{epo}(e_{vpa}(t_i, vp_j)).distCC(X(t_i), Y_t(vp_j)). \quad (5.5)$$

Equation 5.5, shows that C_{Cost} value can be reduced by decreasing the Manhattan distance $distCC(X(t_i), Y_t(vp_j))$ (or hop-to-hop count). For chip multiprocessors having larger core count, efficient mapping and hybrid interconnect can reduce $distCC(X(t_i), Y_t(vp_j))$ significantly which in turn reduce the on-chip power consumption.

Our main aim in this chapter is to map the virtual pages to the memory slices (DRAM with SRAM buffer) dynamically at the run-time so that memory access latency after the last level cache along with the on-chip communication can be minimized, and therefore total execution time of the application and chip power consumption get minimized.

5.2 Self-adaptive Run-time Page Mapping

Consider, the virtual page $vp_j \in \mathbf{VP}$ mapped to a physical page $pp_j \in \mathbf{PP}$ associated to the DRAM slice $m_i \in \{m_0, m_1, \cdots, m_{M-1}\}$. If access request to a cache block (associated to page vp_j) from a thread $t_i \in \mathbf{T}$ get an L2 cache miss, then the processor sends cache block access request (associated to the virtual page address of the page vp_j) to the MC of the respective DRAM slice. The processor gets the information about MC for the page vp_j from the local TLB of the processor using its virtual address. Moreover, virtual page means complete virtual page address and is used interchangeably throughout the thesis.

For each such L2 cache miss associated to pair (t_i, vp_j), this Section presents the adaptive run-time page mapping on to the target CMP system having 3D-stacked DRAM memory. We have considered phase-wise behavior of the multi-threaded application to perform the mapping. To achieve our goal, we have modified the architecture of the memory controller unit associated with each 3D-stacked DRAM slice. The modified architecture of the memory controller (MC) for the target 3D-stacked DRAM memory based CMP system is shown in Fig. 5.1. It has three extra hardware units, specifically (a) page access unit, (b) profiling unit, and (c) page placement unit. The working of the modified MC can be divided into two parts, namely: (a) page

TH-2297_136201001

access, and run-time profiling and, (b) page mapping, migration and TLB update. The following Sub-sections 5.2.1 and 5.2.2 explains the working in detail.

5.2.1 Page Access and Run-Time Profiling

Algorithm 4 shows the pseudocode of the steps associated with the page access and run-time profiling mechanism at each MC.

Page Access: To serve each L2 cache miss associated to pair (t_i, vp_j), at the current MC, migration controller of the **page access unit** checks the status of the page in the migrated list of the MC. The migrated page list can be easily implemented by an array of one-bit Boolean value for each active virtual pages and represented as $Mig[vp_j]$ array. The true value of $Mig[vp_j]$ is interpreted as the page vp_j is migrated to some other DRAM memory slice. Therefore, migration controller forwards such access request ($Mig[vp_j] = 1$) to another appropriate DRAM memory slice, with the target DRAM slice number given by $Loc[vp_j]$ (detail of calculation of the $Loc[vp_j]$ is given in next Sub-section).

Further, if $Mig[vp_j] = 0$, then page access request goes to the corresponding MC associated with the memory slice. So, if the page is present in the DRAM slice, then the access request is served. Otherwise, MC sends a not-found message to the requesting core. The requesting core needs to update its TLB based on the current value of the global page table, and the core initiates re-request.

Run-time Profiling: In parallel with page access mechanism, for each L2 cache miss associated to pair (t_i, vp_j) run-time profiling is also performed by the **profiling unit**. We have used two groups of the profiling counters namely (a) group of the current epoch profile-counters $\mathbf{C_{cntr}}$, and (b) group of the previous epoch profile-counters $\mathbf{P_{cntr}}$, to profile the page access information. All the current epoch profile-counters C_{SNum} (where, $C_{SNum} \in \mathbf{C_{cntr}}$, $\forall\ SNum \in \{0, ..., M\text{-}1\}$) associated with the (vp, pp) and $SNum$ (derived from CST array) are incremented for each access request pair (c_i, vp_j) at the respective memory controller. Core to slice table (CST) gives the value of the 3D-stacked memory slice number $SNum$ for each core. The value $SNum$ is corresponding to the 3D-stacked memory slice number which is closely associated (or local) to the core indexed by 1D-array (further details of CST array is explained in Section 3.1 of Chapter 3).

TH-2297_136201001

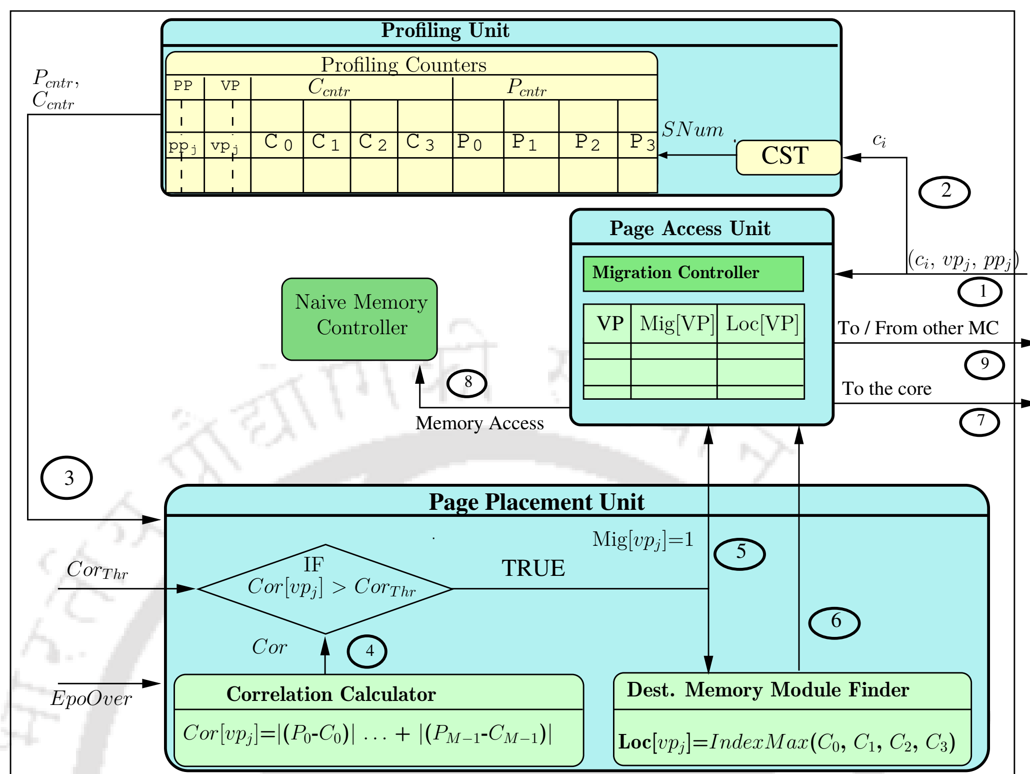


Figure 5.1: Memory controller (naive MC with new mapping hardware)

Algorithm 4 :Page Access and Run-time Profiling

1: **if** $Mig[vp_j]$ is set for vp_j (marked as migrated) **then**
2:    Forwards this access request to $Loc[vp_j]$ DRAM slice.
3:    Finish the access request from the current MC for vp_j.
4: **else**
5:    **if** Page is found in DRAM memory slice **then**
6:       Serve the request from DRAM memory slice.
7:    **else**
8:       MC sends not-found message to the requesting core.
9:       Requesting core updates TLB based on current global page table value.
10:       Re-initialize the access request based on newer TLB value.
11:    **end if**
12: **end if**
13: Profile counters $\mathbf{C_{cntr}}$ is incremented.

5.2.2 Page Mapping, Migration and TLB Update

In this Sub-section, we describe the working of the **page placement unit** to decide on the mapping of a page vp_j at the end of an epoch. Furthermore, we explain page migration and TLB update mechanism. Algorithm 5 shows the steps that need to be performed at the end of each phase (or epoch).

TH-2297_136201001

Algorithm 5 :Page Mapping Decision Making

1: On getting *EpoOver* trigger pulse (*EpoOver* = 1)
2: **for** Each active page vp_j **do**
3: Calculate $Cor[vp_j]$ using Equation 5.6.
4: **if** $Cor[vp_i] > Cor_{Thr}$ **then**
5: Target DRAM slice is calculated using $Loc[vp_j]$.
6: Initiate page migration for vp_j
7: After the end of page migration, set $Mig[vp_j] = 1$.
8: Update the global page table for page vp_j as per the calculated $Loc[vp_j]$ value.
9: **end if**
10: **end for**
11: $\mathbf{P_{cntr}}$=$\mathbf{C_{cntr}}$ and $\mathbf{C_{cntr}}$=0;

Page Mapping: At the end of each phase (or epoch) the *EpoOver* get triggered by pulse and the page placement process get initiated. The page placement unit takes the decision about the mapping of a page vp_j. For each active page (or frame) vp_j (for active page vp_j, $\omega(e_{vpa}(t_i, vp_j)) \neq 0$ for at least one $t_i \in \mathbf{T}$) of the DRAM memory slice, the page placement unit starts taking the decision about the mapping for the page vp_j.

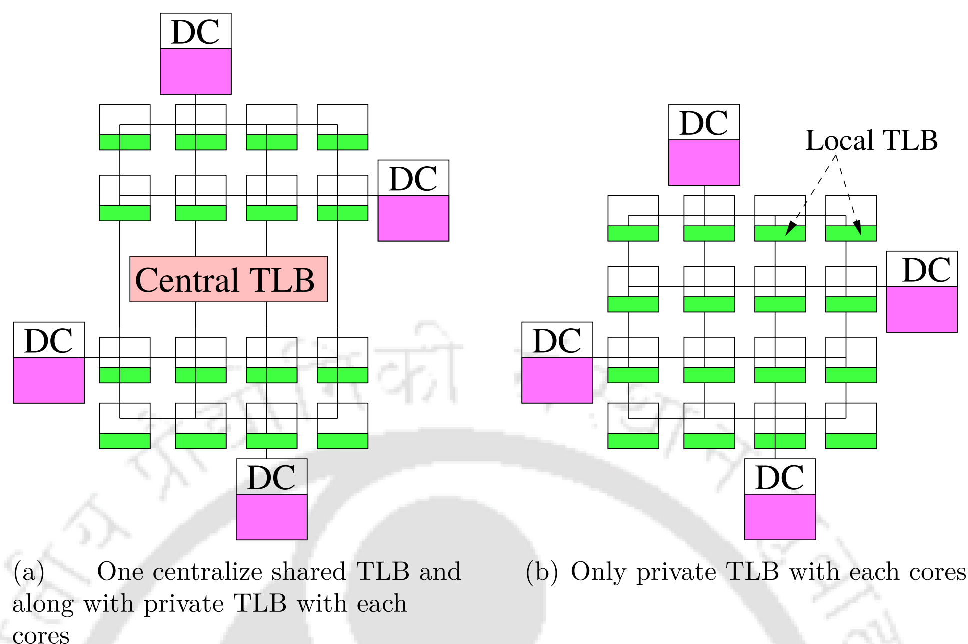
Mapping decision of an active virtual page vp_j is performed by calculating the correlation value $Cor[vp_j]$ among the values of current epoch profiling counter C_{cntr} and values of the previous epoch profiling counter P_{cntr}. Equation 5.6 is used to calculate the value of $Cor[vp_j]$.

$$Cor[vp_j] = |(P_0 - C_0)| + \cdots + |(P_{M-1} - C_{M-1})|, \tag{5.6}$$

where, $C_{SNum} \in \mathbf{C_{cntr}}$ and $P_{SNum} \in \mathbf{P_{cntr}}$ for $SNum \in$ {0, 1, ..., (M-1)} and a page (vp_j, pp_j), as shown in Fig. 5.1.

For any virtual page vp_j, migration decision is calculated by answering the question $Cor[vp_j] > Cor_{Thr}$, where Cor_{Thr} is a parameter value. Cor_{Thr} is used to select the pages for the mapping that are having a larger effect (higher $Cor[vp_j]$ values) on to the latency and avoiding the migration of the less effective pages. Therefore, for a virtual page vp_j, if vp_j is an active virtual page and $Cor[vp_j] > Cor_{Thr}$ is "TRUE" then the target DRAM slice for that page vp_j is calculated. The value of target DRAM slice $Loc[vp_j]$ for a page vp_j is calculated by using $IndexMax(C_{SNum}$ $| \ \forall \ SNum \in$ {0, ..., M-1}), which gives the index $SNum$ associated to the DRAM

TH-2297_136201001



(a) One centralize shared TLB and along with private TLB with each cores

(b) Only private TLB with each cores

Figure 5.2: Shared centralized TLB and distributed private TLB organization of a 4×4 system

slice having maximum C_{SNum} value. The cores covered by this DRAM slice together requested the maximum number of access request (LLC misses) to the page vp_j in the current time epoch.

Page Migration: Further, page migration for vp_j is initiated with the calculated target DRAM slice value $Loc[vp_j]$. It takes time to migrate a page of size about 4KB from one DRAM slice to other, so we have used grace-full page migration (or live page migration). In live page migration, access to the page from source DRAM slice is not stopped immediately when it is being migrated to destination DRAM slice. Page access to the requested page can be served from the destination DRAM slice only after the completion of the migration. Therefore, once the page vp_j is completely migrated to destination DRAM slice, this page is marked as migrated by setting $Mig[vp_j]$=1.

TLB Update Mechanism: After each phase and following the page migration, the global page table gets modified. Therefore, TLB needs to be updated as default TLB update algorithm does not work. There are two ways to solve this problem. Figure 5.2 shows two categories of TLB organizations: (a) one centralized shared TLB and along with private TLB with each core and (b) only private TLB with each core.

TH-2297_136201001

Centralized TLB is similar to shared last level TLB, as explained in [72]. In the case of shared central TLB organization, at the beginning of every time epoch, the shared central TLB gets updated by all the MCs. Each MC updates their list of the page to be migrated (or migrated), and all the page migrations occur at one go at the beginning of the epoch.

Whenever there is access to a virtual page from a core for the first time in current epoch time (phase), the request goes to shared central TLB to get the updated page mapping information about the to be access page. After getting the updated page mapping information, the requested core update its local TLB. After that, access to the same page by using the local TLB. If there is a miss in shared centralize TLB, then the core needs to initialize to bring the virtual page from higher memory or disk into the memory slice local to the core and update the local TLB shared centralized TLB and access the page. Any page miss in local TLB needed to go to shared centralized TLB. The list of communications happened in case of shared centralized TLB organization are:

- Every epoch, all the MCs need to update their list of migration (or placement change) in the shared centralized TLB.
- For every access to a virtual page from the core for the first time in the current epoch, it needs to to get the updated information from shared centralized TLB and access the page from the required DRAM slice.
- All page miss at local processor need to go to Centralized TLB and if there is miss in centralized TLB, the page needed to be brought from secondary storage (or hard disk) and be updated in centralized TLB and local TLB.

Area cost of the shared centralized TLB is higher, as in this method of TLB, we need to store most of the page table or union of all the local TLBs. Also, it needs to update all the updated information in every time epoch.

In the case of *only private TLBs*, the updated information about the pages remapping (or migration), need to send to each core, so that each core can update their local TLB. This update can be done in either immediately at the beginning of the time epoch or in a lazy manner based on the on-demand approach.

- Broadcast Method: In this approach, at the start of every time epoch, each MC send their list of page migration to all the cores. If we assume the number

TH-2297_136201001

of page migrations in an epoch to be limited by a threshold (small numbers), then all the page migrations from the MC can be clubbed together and send the single update message to each core. Otherwise one can use the directory-based TLB update, where update information gets send from MC to cores selectively [72].

- Lazy Method: In this approach, at the beginning of every time epoch, each MC update their list and create a to be migrated list. It does not update to any core immediately. Whenever a page request comes to the MC from a core, if the requested page is migrated to other memory slices, then it forwards this page access request to appropriate MC based on the global page table entry (that got updated after a migration). Moreover, after receiving the page from forwarded MC, the core updates his local TLB. For simplicity, we have considered a lazy method to update the TLB.

Whenever there is a local TLB miss at some core, the core needs to get information about the availability of the virtual page from the global page table, as page table has latest updated page entry. Also, if there is no page table entry found in the page table, then page fault get triggered, and the page entry is brought to nearest DRAM slice from external secondary memory, and page table gets updated accordingly.

In case of the physically addressed cache, tags, as well as cache blocks, need to be updated if the pages get migrated from one memory location to another. Therefore, to enable the run-time page mapping and their migrations, we have considered virtually addresses caches instead of physically addressed.

5.2.3 Experimental Setup

In this work, we have used the Sniper simulator[23] platform to configure the considered target chip multiprocessor architecture. The architecture configuration used is 64 core (8×8) CMP system with four MC, each controlling 1GB of DRAM. Moreover, our method is also applicable to larger CMP systems with their optimal (can be decided using [90, 5]) number of MCs and placements. Each simulated processor is configured with Intel Xeon X550 Gainestown micro-architecture, and other configuration parameters are shown in Table 5.1. Moreover, we modeled the MC, associated hardware, and performance overheads inside the simulator.

TH-2297_136201001

Table 5.1: System configuration parameters

Parameters	Values
Number of tiles	64
Number of cores per tile	1
Frequency of each core	2.66GHz
Number of threads per core	1
Overall 3D-stacked DRAM	4GB, 1GB per DC
DRAM directory type	Full map
Cache block size	64 bytes
Coherence protocol at L2 Level	MESI
Mapping-buffer access latency	24 CPU cycles
DRAM access latency L_{DRAM}	410 CPU cycles
Hop-to-hop latency L_{H2H}	18 CPU cycles
Value for Cor_{Thr}	75

For evaluation, we have used workloads from the multi-threaded PARSEC [14] and SPLASH-2 [118] benchmark suits. Workloads from the PARSEC [14] and SPLASH-2 [118] benchmark suits which generate a higher number of last level cache misses (private L2 cache in our case) are used for the performance evaluation. We run the multi-threaded workloads with *simlarge* input for 10^{10} instruction count.

For simplicity, we have considered the duration of each epoch as 10^7 cycles to trigger the page mapping evaluation. Epoch value 10^7 is chosen for experimental evaluation as the larger value may omit some phases and latency reduction benefits of page mapping associated with these phases. Moreover, a lower value may result in no page migrations and can cause additional latency overhead needed for the evaluation. Our work is not restricted to any specific phase detection technique. Our considered target architecture along with the adaptive run-time page mapping, can work with any good phase detection technique, as given in [29, 11].

5.2.4 Results

Figure 5.3 shows normalized communication cost for different benchmarks using run-time dynamic virtual page mapping to DRAM slices on 4×4 CMP system and we got similar results for 6×6 and 8×8 CMP system. Normalized communication cost for benchmarks without mapping (referred as "before") is 100 for all the benchmarks. With run-time dynamic mapping, the normalized communication cost reduction is 80%, 82%, 83% for ferret benchmark with correlation threshold value 100, 50, and

TH-2297_136201001

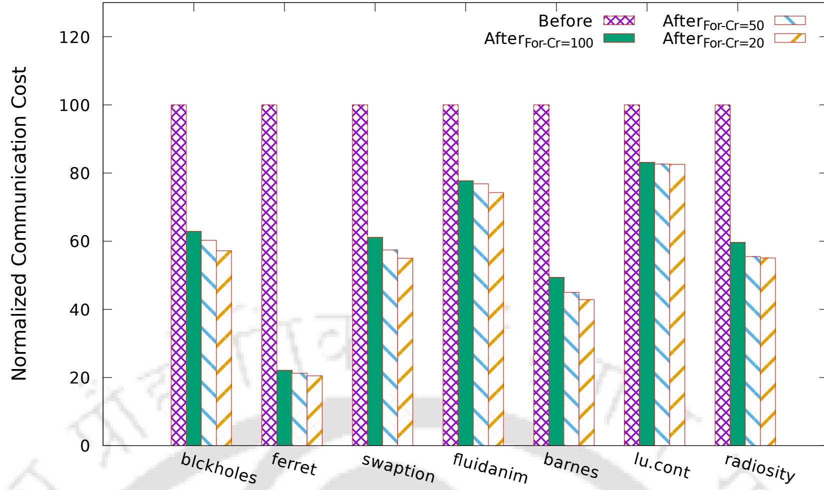


Figure 5.3: Dynamic run-time page mapping considering 4 × 4 CMP

Bench-marks	Naive CCost	Opt CCost	Num-Migr	Migr CCost	STLB CCost	Bcast CCost	Lazy CCost
lu.cont	9.871×10^9	8.324×10^9	64	18384	6184	30920	7748
Omp-fft	1.514×10^7	0.703×10^7	43	11008	2272	11360	2848
Cilk-heat	3.013×10^{10}	1.674×10^{10}	169	43264	7176	35880	9004
Blackholes	2.612×10^9	0.824×10^9	105	26880	2600	13000	3258
swaptions	1.013×10^{11}	0.510×10^{11}	81	20736	3360	16800	4210

Table 5.2: Communication cost (CCost) overhead of different TLB organization and policies for 4 × 4 2D mesh, due to respective operation

20, respectively. Run-time dynamic page mapping results in the reduction of memory access cost by 40%, 80%, 42%, 24%, 52%, 18% and 42% for blckschole, ferret, swaption, fluidanimate, barnes, lucount, and radiosity respectively. Also, improvement using lower correlation threshold is not impressive. So correlation threshold of 100 is sufficient, and we do not need to go for correlation threshold values 20 and 50. The lower value of correlation threshold results in more migrations, but they may not be useful to the reduction of overall on-chip communication cost.

Table 5.2 shows the communication cost overhead of different TLB organization and policies. Column two and three shows overall communication cost due to page access with default static mapping (without page mapping optimization) and with run-time dynamic page mapping to memory slice optimization, respectively. Column 4 shows the number of page migration happened for the benchmarks.

Column 5 shows the worst-case page migration cost for the applications. Page

TH-2297_136201001

migration cost is also less than 0.01% of the overall on-chip communication cost due to page access. Column 6, 7, and 8 show the communication cost overhead of applications for shared centralized TLB, the broadcast approach of TLB update with only local TLB organization, and lazy (or on-demand) approach of TLB update with only local TLB organization respectively. We can see that the communication cost overhead for application is less than 0.01% of overall communication cost due to page access. So any method of TLB organization works for the smaller number of page migrations (when the threshold on correlation is higher even if the overall communication cost reduction is above 50%). Shared centralized TLB require extra hardware to store at least the union of all the local TLB of cores.

5.3 Comparison with Coherent DRAM Cache

In addition to the communication cost C_{Cost} analysis in the previous Section 5.2, in this Section we perform a comparative study between our proposed self-adaptive run-time page mapping and a recent state-of-art work proposed by Chou *et al.* [30]. This comparative study is performed to analyze the effectiveness of the proposed self-adaptive run-time page mapping. Overall application execution time is used as the performance metric to perform the comparative study in this Section.

In [30], authors have proposed a framework called **CANDY** and used the 3D-stacked DRAM memory as a coherence cache in the CMP system to analyze the speedup of the system. Further, to avoid the coherence directory access related overheads for the large DRAM cache, they have proposed a technique that uses on-chip SRAM based buffer to store the frequently accessed cache directory information. Therefore, we have considered the on-chip SRAM based buffer and re-proposed to use it as a cache of the DRAM memory slice, while performing the page mapping. Each SRAM based buffer or mapping buffer ($Mbuff$) stores (or caches) *the frequently accessed cache blocks* of the pages associated with the DRAM memory slices and mitigates the large DRAM access latency. Mapping buffers introduced at every individual DRAM memory slice is having a single and distinct copy of the cache blocks. There is no two mapping buffers (associated to two different 3D-stacked memory slices) that can have same copy of a cache block. Therefore, each mapping buffer is having single and distinct cache block stored in it.

Moreover, the coherence maintenance is used at the private last level processor

TH-2297_136201001

caches, and we have used MESI cache coherence protocol to maintain the consistency. Also, the size of overall last level cache is in few MBs; so the size of coherence directory to maintain the consistency is very small (in few kb).

To achieve the goal of this Section, we have modified the self-adaptive run-time page mapping technique by placing the $Mbuff$ into the page access unit of the modified MC. Fig. 5.4 shows the modified page access unit, which is having $Mbuff$ in it. Therefore, the working of the modified MC associated to the modified self-adaptive run-time page mapping which considers the $Mbuff$ is exactly same as described in Section 5.2 except the changes in the page access unit.

Therefore, for an access request corresponding to the (t_i,vp_j) pair at the MC, if the modified page access unit founds that the $Mig[vp_j]$=0 then $Mbuff$ is checked to get the corresponding cache block entry (associated to page vp_j). So, if the entry is found in $Mbuff$ (a hit in $Mbuff$), then the latency to access the page is less as it is an SRAM buffer access. On the other hand, if the entry is not found in $Mbuff$ (a miss in $Mbuff$), then the page access request goes to the corresponding DRAM slice. After serving the request from DRAM memory slice, associated $Mbuff$ is get updated for the future references. The latency to access the page in case of $Mbuff$ miss is the sum of SRAM buffer access latency and DRAM access latency.

To mitigate the large DRAM memory access latency, we require a high hit rate for the $Mbuff$. The hit rate of $Mbuff$ mainly depends on its interaction with DRAM memory for the page request and its size. For simplicity, we have used a temporal locality-based technique to insert a cache block into $Mbuff$ and maximize the hit rate. To exploit temporal locality, we have used the LRU algorithm for cache block insertion into the $Mbuff$. As most of the cache block access requests (supposed to be DRAM access) get served by the $Mbuff$, so the number of access to DRAM memory reduces which reduces the overall access latency.

We have used a full associative $Mbuff$ that uses the Least Recently Used (LRU) algorithm. Whenever a cache block is referred and is not present in $Mbuff$, it is fetched from DRAM and inserted using the LRU algorithm. However, searching full associative $Mbuff$ for a referred cache block is time-consuming. In this case, we do not use set-associative $Mbuff$ which may degrade the performance by replacing a cache block that is least recently used in a particular set and not from the whole $Mbuff$. Therefore, we have used full associative $Mbuff$ with the facility to look-up 16 cache blocks at a time in parallel while searching for a cache block. Although

TH-2297_136201001

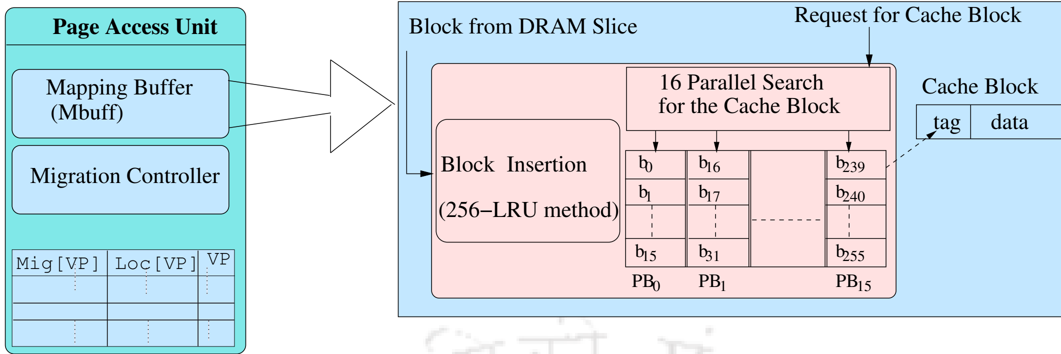


Figure 5.4: Overview of 16KB $Mbuff$

searching more cache blocks in parallel makes $Mbuff$ look-up time small, but it incurs some hardware overhead (in terms of the parallel comparator circuit). Therefore, we have used searching 16 entries at a time for faster look-up while incurring hardware overhead of only 16 comparators. Fig. 5.4, shows an example of 16KB $Mbuff$. It is having facility to look-up 16 blocks in parallel (this parallel blocks are PB_0, PB_1, $\cdots$, PB_{15}) at a time while looking for a cache block entry. For the size of 64 bytes cache block, the $Mbuff$ (of 16KB) have space for 256 cache blocks, so the $Mbuff$ uses 256-LRU method to insert a new cache block entry in a full associative fashion.

Also, once a page vp_j is completely migrated to destination DRAM slice after the page mapping, this page is marked as migrated by setting $Mig[vp_j]$=1. Moreover, all the cache block entries associated to page vp_j is flushed out from the current $Mbuff$.

5.3.1 Results

Fig. 5.5 shows the normalized performance evaluation for different benchmarks. Moreover, in Fig. 5.5, "BC", "BC-Mbuff", "DCC", "CANDY-Dbuff", "PMM" and, "PMM-Mbuff" acronyms refers the following:

- **BC (base-case for this work)**: DRAM is used to store only local data of the memory node. Moreover, this method does not consider the run-time page mapping and extra SRAM buffer.

- **BC-Mbuff**: DRAM is used to store the local data along with the use of 64KB SRAM buffer to store the frequently accessed cache blocks. However, this method does not use run-time page mapping.

- **DCC**: DRAM is used as a coherent cache (stores local and remote data) with

TH-2297_136201001

coherence directory within DRAM. However, this method does not use the run-time page mapping and extra SRAM buffer.

- **CANDY-Dbuff**: DRAM is used as a coherent cache (stores local and remote data) along with the SRAM buffer *to store the coherence directory information* (as proposed by Chou *et al.* in [30], and 8MB temporal SRAM buffer) and no run-time page mapping is performed.
- **PMM**: DRAM stores only local data and run-time page mapping are performed without any SRAM buffer.
- **PMM-Mbuff (our method)**: DRAM stores local data along with the use of adaptive run-time page mapping. Also, a 64KB SRAM buffer is used *to store the frequently accessed cache blocks.*

Fig. 5.5 shows the performance comparison considering L_{DRAM}=410 cycles and L_{H2H}=18 cycles. The DRAM access latency and the inter-node latency values are considered based on the real system values [107, 131]. In Fig. 5.5, for the real systems, method “PMM-Mbuff” outperforms “BC” by an average of 48%. “PMM” and “BC-Mbuff” methods improve performance by an average of 6% and 42% respectively when compared to base-case. Moreover, when compared with “DCC” and “CANDY-Dbuff” our method “PMM-Mbuff” shows an average improvement of about 43% and 40% respectively.

Our method “PMM-Mbuff” shows the performance improvement due to, first, the page migrations towards nearest DRAM slice, and this causes a significant reduction in the inter-node latency. Second, the substantial reductions due to the low latency of $Mbuff$ for all the DRAM references. “DCC” also improves an average of about 8% over the base-case. However, due to the DRAM access latency overhead of the larger DRAM based coherence directory and extra invalidation overhead, it does not performs well and some times it may degrade the performance of the benchmark (for example, facesim). “CANDY-Dbuff” uses an SRAM based buffer to reduce the DRAM based coherence directory access latency, and it improves an average of about 10% (for temporal locality method) over base case. However, “CANDY-Dbuff” again possess extra SRAM buffer latency overhead (to access the coherence directory information) which is not present in our method. Moreover, our approach reduces the remote data access latency and DRAM access latency by utilizing the page mapping and $Mbuff$, respectively.

TH-2297_136201001

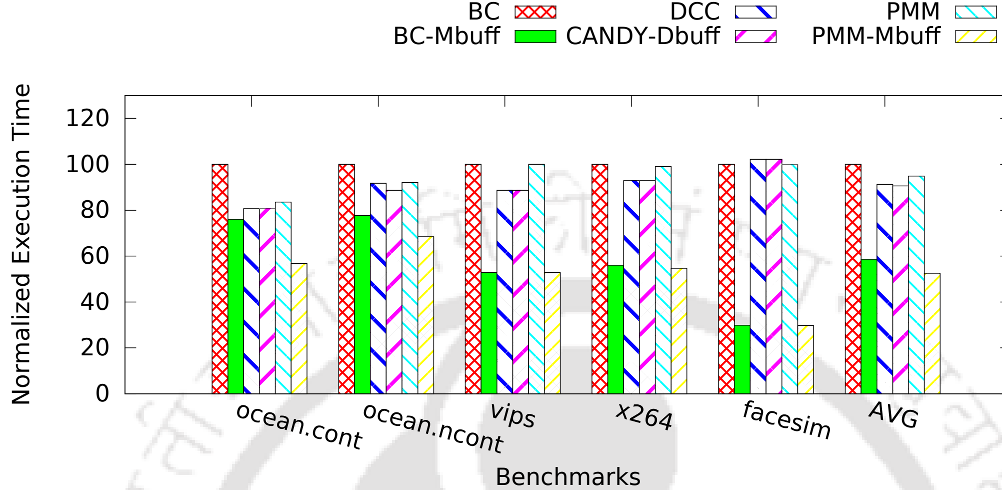


Figure 5.5: Overall normalized execution time for different benchmarks, average is reported as "AVG"

We performed a sensitivity study for the size of $Mbuff$. Fig. 5.6 shows an example of the $Mbuff$ hit rate for its different size (in KB). The larger size of $Mbuff$ is better as it gives a higher hit rate; however, its SRAM based on-die architecture makes its larger size costlier. Therefore, we have used 16KB $Mbuff$ per memory controller, which is giving an average hit rate of about 80% for all the benchmarks.

Note that the use of the SRAM based $Mbuff$ and it's 80% hit rate lowers the average memory access time (AMAT) of the DRAM memory. Consider, the centers of the four non-overlapping areas 17, 13, 50, and 46 as represented in Fig. 3.4, the average hop-to-hop distance is assumed to be the average distance between the DRAM slices. So, for a DRAM slice to any other DRAM slice, the average hop-to-hop distance is 6, which is the average of the Manhattan distances between all the DRAM slices ($6 = \frac{5+5+8}{3}$ based on XY-routing and Fig. 3.4). Therefore, using the latency values from Table 5.1 and 80% $Mbuff$ hit rate, the AMAT of the DRAM memory is 106 cycles ($24 + 0.2 \times 410$) and average hop-to-hop network latency is 108 cycles (6×18). Moreover, without $Mbuff$, the AMAT of the DRAM memory is 410 cycles. Thus, for the considered latency values (as given in Table 5.1), $Mbuff$ lowers the AMAT of the DRAM memory and makes it comparable to the average network latency value. *When average network latency value is comparable or higher to the AMAT of the DRAM memory, our method "PMM-Mbuff" performs better.*

TH-2297_136201001

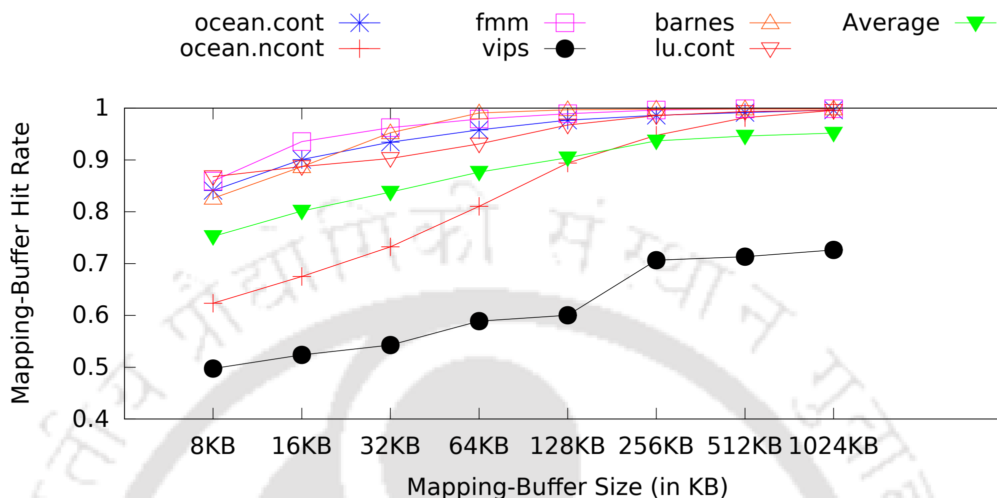


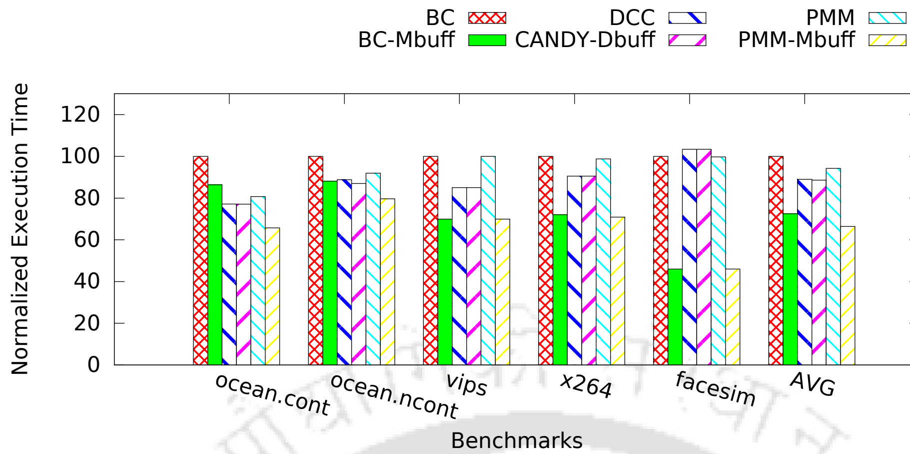
Figure 5.6: Mapping-buffer hit rate for its different size

To analyze the effect of the DRAM access latency L_{DRAM} as well as hop-to-hop latency L_{H2H} values, we performed further experiments considering some variations of L_{DRAM} and L_{H2H} values. Fig. 5.7(a) shows the result associated with the L_{DRAM}=210 cycles and L_{H2H}=18 cycles. Similarly, Fig. 5.7(b) is associated with the L_{DRAM} and L_{H2H} values of 410 cycles and 48 cycles respectively.

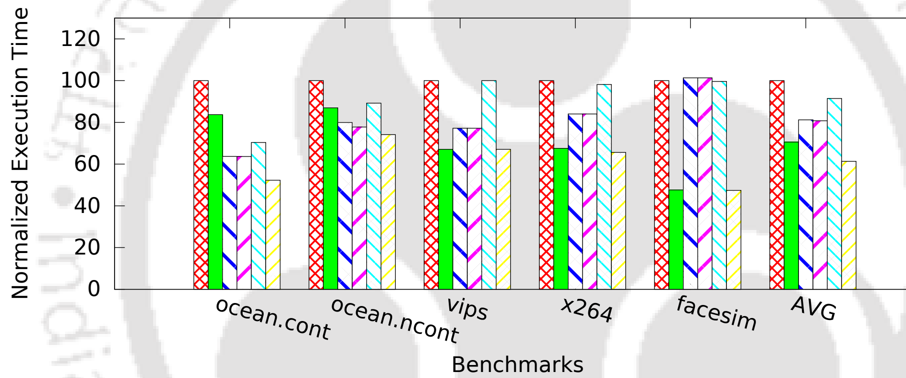
Considering the $Mbuff$ hit rate of 80% and SRAM access latency of 24 cycles, the AMAT value of the DRAM memory is reduced to 106 cycles (24+0.2×410) corresponding to DRAM access latency L_{DRAM} value of 410 cycles, which is about 75% reduction. However, the AMAT value of the DRAM memory is reduced to 66 cycles (24+0.2×210) corresponding to DRAM access latency L_{DRAM} value of 210 cycles. Therefore, benefits of the $Mbuff$ diminishes for the smaller value of the DRAM access latency L_{DRAM} (or faster DRAM), as shown in Fig. 5.7(a) and 5.7(b). In Fig. 5.7(a) and 5.7(b) the method "BC-Mbuff" reduces the execution time by an average of 28% and 30% as compared to the "BC"; whereas, in Fig. 7.3, method "BC-Mbuff" has reduced the execution time by an average of 42%. Moreover, methods "DCC", "CANDY-Dbuff", and "PMM" in Fig. 5.7(a) performs almost similar to the as shown in Fig. 5.5.

Further, Fig. 5.7(b) shows that methods "DCC", "CANDY-Dbuff", and "PMM" reduces the execution time by an average of 19%, 29%, and 10% respectively as

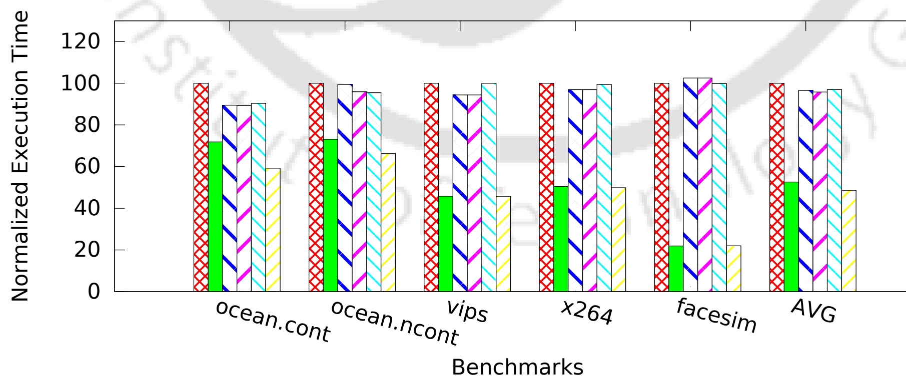
TH-2297_136201001



(a) L_{DRAM}=210 cycles and L_{H2H}=18 cycles



(b) L_{DRAM}=410 cycles and L_{H2H}=48 cycles



(c) L_{DRAM}=410 cycles and L_{H2H}=9 cycles

Figure 5.7: Overall normalized execution time considering different L_{DRAM} and L_{H2H} values, average is reported as "AVG"

TH-2297_136201001

compared to the "BC". This increase in the execution latency reduction as compared to the results shown by Fig. 5.5, is due to the larger dominating hop-to-hop latency L_{H2H} value. Also, it is evident from the result that the methods "DCC", "CANDY-Dbuff", and "PMM" perform better along with the increase in the hop-to-hop latency L_{H2H} value. Whereas, Fig. 5.7(c) shows that for L_{DRAM}=410 cycles and L_{H2H}=9 cycles, the methods "DCC", "CANDY-Dbuff", and "PMM" does not perform good and improves performance by an average of 3.50%, 4.50%, and 3% as compared to the "BC"; whereas, the method "PMM-Mbuff" outperforms the "BC" by an average of 51%.

5.4 Performance and Area Overhead

In DRAM memory, row buffer miss rates (associated to the cache blocks) are usually 65% for the multi-threaded workloads. Therefore, page access latency is almost the same as block access latency as most of the DRAM uses page mode access. Assuming page mode DRAM and have row buffer size $\geq$ 4KB (otherwise, two or more DRAM bank can be used to make the page size $\geq$ 4KB). Therefore, for 4KB page size and 64 bytes block size, each page migration incurs a latency overhead of only two DRAM access latency (one at a source and another at a destination).

In almost all the contemporary CMPs, the standard size of the message (or packet) for NOC is 64 bytes, therefore 64 packets needed to transfer a 4KB page. Moreover, the maximum page migration cost associated with each page is 64 times the maximum distance between the source memory slice and destination memory slices. To avoid the additional network interconnect overhead, we have used the same processor layer network for page migration.

In Table 5.3, column four shows the percentage of overall page migration calculated for the total number of L2 cache misses. It shows that migration due to run-time adaptive page mapping incurs on an average 3.52% overhead as compared to the total number of L2 cache misses. For simplicity and due to the small percentage of the total number of page migration, we have not considered network contention due to page migration. The live page migration proceeds in parallel with the application execution without pausing (or interrupting) it directly, therefore application performance do not get affected by migration overheads (except due to some network traffic). The number of page migration can also be regulated using Cor_{Thr} parameter value.

TH-2297_136201001

Table 5.3: Example: Total number of L2 cache misses and page migration

Benchmarks	**# L2 Misses**	**# Migrations**	**Migration %**
ocean.cont	1.19×10^8	1.66×10^6	1.38%
ocean.ncont	1.79×10^8	5.67×10^6	3.15%
vips	2.99×10^7	1.86×10^5	0.62%
x264	2.98×10^7	1.53×10^5	0.51%
facesim	8.99×10^7	7.11×10^5	0.79%

Given a DRAM memory, consider a 4KB page and 4 DRAM slices (M=4) with an overall size of G_{size}=4GB, there are $\frac{4GB}{4KB}$ number of physical pages. Therefore, we require $\frac{4GB}{4KB} \times 2 \times 4$ number of 16 bit counters (associated with the C and C' group of profiling counters), M+1 comparisons, M subtractions and M additions for the 4GB of DRAM memory. Moreover, for each physical page, it takes $log_2(M)$ cycles to compute using M adder and comparator units. For M=4 number of DRAM slices, the modified MC incurs a hardware overhead of 4 adders and comparator along with the time overhead of 4 cycles. Also, to create the migration list Mig, we need an overall of $\frac{4GB}{4KB}$ bit or 128KB of memory space.

For profiling counters, we need 16MB memory space corresponding to 4GB of DRAM, whereas for coherence directory of DRAM cache we need 256MB space for 64 byte cache block (26 bits for block identification ($\frac{4GB}{64B} = \frac{2^{32}}{2^6} = 2^{26}$ cache blocks), 2 bits to maintain coherence and 2 bits for DRAM slices per cache block) [30]. Moreover, the $Mbuff$ causes an SRAM memory overhead of 64KB (16KB×4, for all the 4 $Mbuff$s). So, area overhead in our case is significantly less as compared to the architecture where DRAM is used as a coherent cache.

Table 5.4, summarizes the area overheads associated with the different components used in the modified DC. In Table 5.4, the first three columns from left show the storage overhead associated with the profiling counters, migration (Mig) list, and location (Loc.) array, respectively. Moreover, column four and five show the total number of flip-flops (FFs) and look-up tables (LUTs) used to design the additional controller components (including adder, comparator, subtractor, and basic finite state machines (FSMs)) of the DC that performs the page mapping. We have used the Xilinx Vivado (version 2014.3.1) high-level synthesis (HLS) tool to estimate the area overhead of the additional components. Also, from Table 5.4, we can see that controller area overhead (associated to the adders, comparators, subtractors, and basic FSMs) are very less as compared to the storage area overhead (associated to the

TH-2297_136201001

Table 5.4: Area overhead summary.

Storage			**Controller (including Add., Sub. and Comp.)**	
Profiling Counters	Mig. List	Loc. Array	# FFs	# LUTs
16MB	128KB	256KB	1207	1820

profiling counters, migration (Mig) list, and location (Loc.) array).

5.5 Summary

In this chapter, we have proposed a self-adaptive run-time page mapping on to the 3D-stacked DRAM memory based CMP system. Further, we have compared our method with a recent state of work as proposed in [30]. Our results and analysis show that the proposed method can be an alternative way to use the 3D-stacked DRAM memory for current as well as future CMP systems.

The proposed self-adaptive run-time page mapping alone shows the communication cost reduction up to a maximum of 80% and an average of about 40% as compared to the base case method. Further, our self-adaptive run-time page mapping together with the SRAM mapping buffer outperforms the base-case by an average of 48% over base-case in terms of overall execution time. Also, most importantly, the adaptive run-time mapping with the SRAM mapping buffer shows a performance improvement by an average of 40% (in terms of overall execution time) when compared to 3D-stacked DRAM used as a coherent cache with temporal SRAM buffer, a state-of-art work proposed by [30].

One important fact is that the most of the applications exhibit phase wise behaviour, but some applications may not have run time phase wise behaviour. For such applications also our mapping approach will work and the proposed dynamic run time mapping will behaves like static mapping 4, where one time optimized mapping will happen.

TH-2297_136201001

TH-2297_136201001

6

Run-time Page Mapping Considering Hybrid Memory

Although 3D-stacked DRAM memory has various advantages, the power consumption of the 3D-stacked DRAM is rapidly growing with the capacity increase. Fig. 6.1 shows that the power consumption due to the DRAM refresh has reached beyond 40% as compared to the overall DRAM power consumption, as reported by Mutlu *et al.* [79]. To tackle the DRAM refresh power consumption overhead associated with the larger size DRAM memory, many non-volatile memories such as PCM, MRAM, ReRAM have been explored as an alternative to the DRAM [18, 125, 129, 99]. Among all types of NVM, PCM memory has been extensively studied by the researchers in combination with the DRAM memory due to its overall better performance and scalability. PCM memory offers higher density and lower leakage power consumption (as there is no row refresh in PCM) as compared to the DRAM memory. However, PCM has some drawbacks as compared to the DRAM memory, for example- higher read/write latency and lower write endurance [124, 63].

The hybrid memory constituted using DRAM and PCM has been proposed as a potential solution to take the benefits of both the technologies [64, 124, 126, 96]. This hybrid memory possesses many benefits such as are lower leakage power, higher density, lower read/write latency, and improved write endurance. In the CMPs where on-chip memory has the capacity in tens of GBs, investigating the impact of data placement becomes crucial. Studies related to the 3D-stacked hybrid memory-based CMPs have mainly considered the cache architecture for these memories [127]. How-

TH-2297_136201001

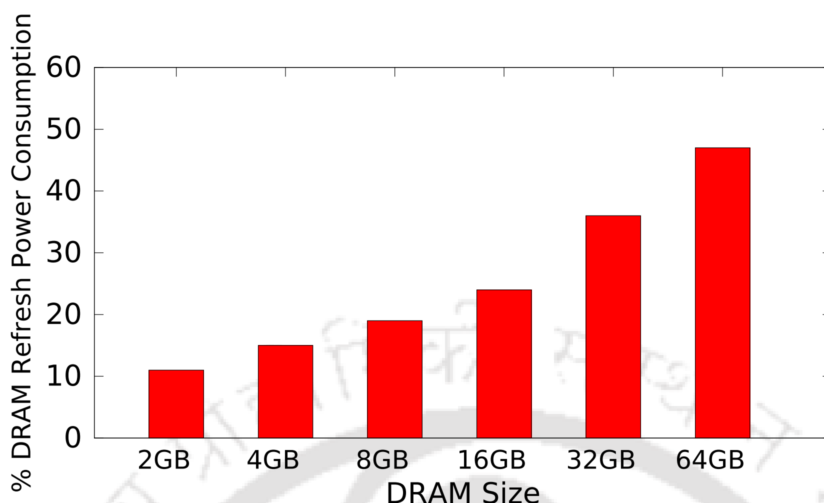


Figure 6.1: DRAM refresh power consumption for DRAM devices [79].

ever, cache architecture requires a large coherence directory to maintain the coherence. This, in turn, causes area as well as access time overheads due to the large size coherence directory. Therefore, to take the benefits of the large capacity 3D-stacked hybrid memories, studies need to consider higher granularity data placement and memory management schemes for these memories.

In this chapter, we have considered non-coherent 3D-stacked hybrid DRAM-PCM memory-based CMP system. Fig. 6.2, shows the logical block diagram of the hybrid memory slice (or hybrid memory module) where DRAM memory works as the cache for the PCM memory and initially entire application working set resides in the PCM memory [95, 124, 64]. This type of architecture is chosen for this work due to its lower management cost and better average performance [93]. Further, to take advantage of the large capacity 3D-stacked hybrid memory and to minimize the remote memory accesses overheads, we have performed an access-aware self-adaptive run-time page mapping.

Specifically, in this chapter, we make the following contributions by performing the architectural changes in each memory controller unit.

- We have designed a simple DRAM access-aware run-time page placement technique between DRAM and PCM of the hybrid memory to reduce the DRAM refresh operations and its associated power consumption overhead.
- Further, based on the DRAM row access information, we performed an access-aware self-adaptive page mapping for the optimized page placement between

TH-2297_136201001

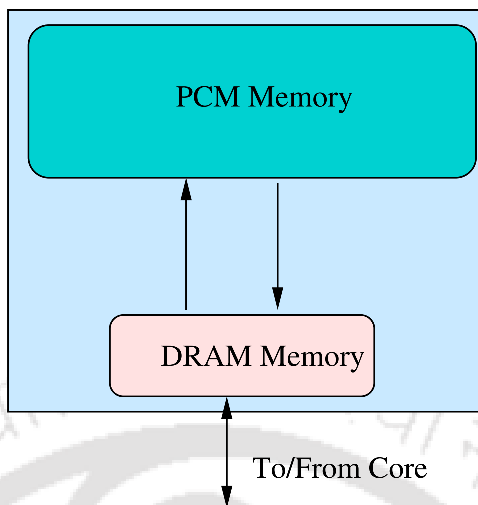


Figure 6.2: Hybrid memory module architecture

the different hybrid memory modules of the 3D-stacked hybrid memory.

The working of our proposed technique is classified into two parts, (a) *access-aware self-adaptive page mapping on to the hybrid memory slices*, and (b) *DRAM access-aware page placement between DRAM and PCM memory of the hybrid memory slice*.

To perform (a), we have considered the phase-wise behavior of the applications, and for simplicity, we assume that the run-time of an application is divided into multiple phases (or epochs) of fixed length. We have used *EpoOver* to trigger the end of a phase (or epoch), and perform the access-aware self-adaptive page mapping on to hybrid memory slices. Similarly, to perform (b), we have considered that the DRAM row gets decayed and inactive due to the leaky nature of its constituent cells if the rows did not get access for a longer time [93] and need a refresh to retain the information within it. Therefore, to eliminate the refresh operations associated with the decayed and inactive rows, we perform row-level periodic monitoring of the DRAM rows. Moreover, based on the last access time, we evict the decayed rows from the DRAM to PCM at the end of the row monitoring period. We have used, *EndPeriod* to trigger the end of the row monitoring period and to start the DRAM access-aware page placement decision between DRAM and PCM memory of the hybrid memory slice.

In this chapter, the value of an epoch (or phase) and row monitoring period are selected such that we can perform multiple row monitoring within an epoch to give more priority to the page migration within a hybrid memory slice (between DRAM

TH-2297_136201001

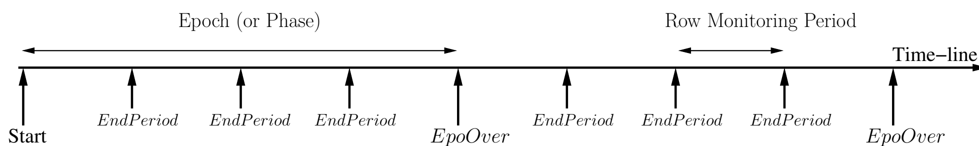


Figure 6.3: Relation between an epoch and row monitoring period

and PCM memory). As the row monitoring period is having the value of 128ms (or 3.44×10^8 cycles); therefore, the fixed-size epoch value is chosen as 10^9 cycle. As an example, Fig. 6.3 shows the relation between the epoch size and the row monitoring period size.

Following sections 6.1 and 6.2 explains the working of (a) access-aware self-adaptive page mapping on to the hybrid memory slices and (b) access-aware page placement between DRAM and PCM of the hybrid memory slice respectively.

6.1 Access-Aware Self-adaptive Page Mapping on to Hybrid Memory Slices

Access-aware self-adaptive page mapping is similar to the self-adaptive run-time page mapping as proposed in section 5.2 of chapter 5, except for some essential modifications. However, for completeness, we brief the self-adaptive run-time page mapping (as proposed in section 5.2) and avoid the details that are already explained in section 5.2.

To perform the access-aware self-adaptive page mapping on to hybrid memory slices at the application run-time, we have modified the architecture of the memory controllers. Fig. 6.4 shows the modified MC for the 8×8 CMP system with four memory slices. Consider, a virtual page vp_j is mapped to a physical page or frame pp_j associated with the hybrid memory module $m_i \in \{m_0, m_1, m_2, m_3\}$. Therefore, for each L2 cache miss generated from the core c_i associated to the page vp_j, the working of the modified MC can be categorized into two parts, namely, (a) page access and run-time profiling, and (b) page mapping decision making. Following subsections, 6.1.1 and 6.1.2 explains the working mechanism in detail.

TH-2297_136201001

6.1.1 Page Access and Run-Time Profiling

Algorithm 6 shows the steps needed to perform for the page access and run-time profiling mechanism at each memory controller, which is similar to as described in Sub-section 5.2.1 of Chapter 5 except the DRAM to PCM page migration-related mechanisms. So, as described in previous Chapter 5, for each access request pair (c_i, vp_j) associated to the L2 cache misses ①, **migration controller** checks the status of the migrated list value $Mig[vp_j]$ in the Mig. Migrated list Mig is a one-dimensional array at an MC, and it stores a set value for a page migrated to any other hybrid memory module (or hybrid memory slice). Set value of $Mig[vp_j]$ in the migrated list Mig interprets that the page vp_j is migrated to some other hybrid memory module. Therefore, the migration controller forwards this access request to the appropriate hybrid memory module (decided by the location array $Loc[vp_j]$) ⑨.

Whereas, a not set value of $Mig[vp_j]$ determines that the requested entry is available in the current hybrid memory module and such request goes to the DRAM refresh controller ⑧. DRAM refresh controller at the current MC checks the presence of the page vp_j in the DRAM memory. If the page is not present in the DRAM memory, then the page is brought from the PCM memory to serve the core. This page is inserted into an appropriate row of the DRAM memory using the least-frequently-used (LFU) technique. Moreover, if the page is not present in the PCM, then MC sends a not-found message to the requesting core. The requesting core needs to update its TLB based on the current value of the global page table, and the core initiates re-request. Also, page fault in the global page table is dealt as per the default method. In parallel, the core access request is served along with the initiation of a TLB update message. As, a page is brought from the PCM memory to DRAM memory, so page table entry gets modified. Therefore, the TLB of the requesting core needs to be updated accordingly. Along with all these necessary measures, a standard DRAM to PCM write-back is made if required. Therefore, for the access request to the page vp_j, valid bit $ValB[vp_j]$ is set to “TRUE” for the corresponding DRAM row entry. Also, for the row and their associated mapped page vp_j, the access bit array $AccB[vp_j]$ and dirty bit array $DirtB[vp_j]$ status is maintained. Arrays $ValB$, and $DirtB$ are used to keep track the validity (valid or invalid DRAM row), and dirty status respectively to each DRAM row and their associated pages in the DRAM memory of a hybrid memory slice. Also, $AccB$ array is used to store the access status of each DRAM row. While serving to a core c_i ⑦, if the page is found then the access

TH-2297_136201001

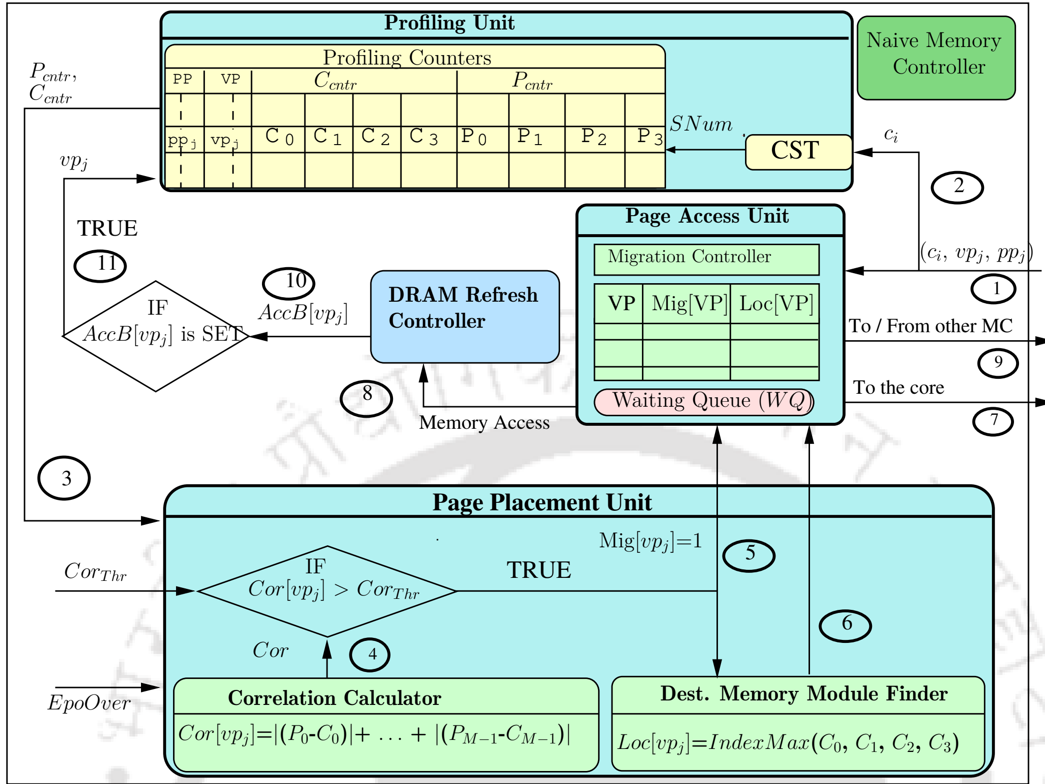


Figure 6.4: Memory controller (naive MC with new mapping hardware)

bit array $AccB[vp_j]$ is set to 'TRUE' upon each such access and reset to 'FALSE' at the decision point (described in section 6.2).

In parallel with the page access mechanism and to decide on the page mapping, for each pair (c_i, vp_j) run-time profiling is performed by the **profiling unit**. We have used two groups of the profiling counters namely (a) group of the current epoch profile-counters C_{cntr}, and (b) group of the previous epoch profile-counters P_{cntr}, to profile the page access information. All the current epoch profile-counters C_{SNum} (where, $C_{SNum} \in C_{cntr}$, $\forall$ $SNum \in \{0, ..., M\text{-}1\}$) associated with the vp and $SNum$ (derived from CST array) are incremented for each access request pair (c_i, vp_j) at the respective memory controller. Core to slice table (CST) gives the value of the 3D-stacked hybrid memory slice number $SNum$ for each core. The value $SNum$ is corresponding to the 3D-stacked hybrid memory slice number which is closely associated (or local) to the core indexed by 1D-array (details of CST array is explained in section 3.1 of chapter 3).

TH-2297_136201001

Algorithm 6 :Page Access and Run-time Profiling for Hybrid Memory

1: **for** Each memory access associated to the page vp_j **do**
2:   **if** $Mig[vp_j]$ is set for vp_j (marked as migrated) **then**
3:     Forwards this request to the hybrid memory module calculated by $Loc[vp_j]$.
4:   **else**
5:     **if** vp_j is not in DRAM **then**
6:       **if** vp_j is in PCM **then**
7:         Bring vp_j from PCM to DRAM Page Buffer.
8:         Serve the requesting core c_i along with the TLB update mechanism, and
9:         Place it into a DRAM row.
10:         Perform DRAM to PCM write-back if needed and flush the associated counters.
11:       **else**
12:         MC sends not-found message to the requesting core.
13:         Requesting core updates TLB based on current global page table value.
14:         Re-initialize the access request based on newer TLB value.
15:       **end if**
16:     **end if**
17:     In parallel set the $ValB[vp_j]$ =TRUE,
18:     Set the access array $AccB[vp_j]$ =TRUE, and
19:     **if** vp_j is write operation **then**
20:       Set dirty array $DirtB[vp_j]$ =TRUE.
21:     **end if**
22:   **end if**
23:   Profile counters (C_{cntr}) is incremented.
24: **end for**

6.1.2 Page Mapping Decision Making

After the access information is profiled throughout an epoch, at the end of the phase (or epoch), modified memory controller decides the mapping of the pages residing (or mapped) only to the DRAM memory of the current hybrid memory module. Thus, as opposed to the previous Chapter 5 where all the active pages of a memory module (or memory slice) are checked for the mapping decision, in this Chapter we consider only those pages that reside in the DRAM memory of the hybrid memory module for the mapping decision. Algorithm 7 shows the steps to be performed at the end of each epoch for the mapping decisions.

Page placement unit activates the end of a phase (or an epoch) to start taking the mapping decision about the pages residing in the DRAM memory of the hybrid

TH-2297_136201001

Algorithm 7 :Page Mapping Decision Making for Hybrid Memory

1: **if** $EpoOver$=TRUE **then**
2: **for** Each page vp_j in the DRAM memory slice of the current memory module **do**
3: **if** $AccB[vp_j]$= TRUE for the page vp_j **then**
4: Calculate $Cor[vp_j]$ (using Equation 5.6).
5: **if** $Cor[vp_i] > Cor_{Thr}$ **then**
6: Calculated target hybrid memory module position using $Loc[vp_j]$.
7: Initiate the page migration for the page vp_j.
8: Once, migration is completed set the $Mig[vp_j]$=TRUE.
9: Update the global page table for page vp_j as per the calculated $Loc[vp_j]$ value.
10: **else**
11: Page vp_j remain in the current DRAM only.
12: **end if**
13: **end if**
14: **end for**
15: **end if**
16: P_{cntr}=C_{cntr} and C_{cntr}=0;

memory slices by triggering the $EpoOver$.

For each recently accessed page vp_j having $AccB[vp_j]$ as the "TRUE" value, the page placement unit starts deciding about their mapping. A recently accessed page does not need to be refreshed as it is recently got refreshed or activated by an access request at the DRAM memory. Therefore, mapping decision considers the pages that are residing in the DRAM memory of the hybrid memory slices. The value of the access bit array $AccB[vp_j]$, corresponding to the page vp_j is decided by the DRAM refresh controller.

Mapping decision of a recently accessed page vp_j (which is a page that resides in the DRAM of the hybrid memory module) is performed based on the correlation value $Cor[vp_j]$ between the profile-counters C_{cntr} and P_{cntr} ③, which is similar to as explained in Section 5.2.2. Therefore, for a page vp_j if the access bit array value $AccB[vp_j]$ is set ⑪, ⑩, only then **correlation calculator** calculates the correlation value $Cor[vp_j]$ using Equation 5.6 and similar to as explained in Section 5.2.2 of Chapter 5.

Therefore, for a recently accessed page vp_j, if $Cor[vp_j] > Cor_{Thr}$ [1] is "TRUE" ⑤, then page vp_j becomes the candidate for the migration. Further, the target hybrid

[1]The Cor_{Thr} is a parameter value and is used to select the pages for the mapping that are having a larger effect (larger $Cor[vp_j]$ values) to the system latency. Details are given in Section 5.2.2

TH-2297_136201001

memory slice value $Loc[vp_j]$ for the page vp_j is calculated by the **destination hybrid memory module finder** ⑥, similar to as explained in Section 5.2.2 of Chapter 5. Also, each page whose migration is initiated, wait in the waiting queue WQ to get updated into the migration list Mig till the completion of the migration.

Page migration: Once, the target memory slice value $Loc[vp_j]$ is calculated, migration of the pages are initiated one by one and distributed over the entire epoch time to avoid sudden traffic overhead ⑨. Migrating a page (size about 4KB) from one place to other incurs time overhead; therefore, this work uses a lazy page migration technique. In this lazy page migration technique, we do not stop the access to a page from the source hybrid memory module unless this page gets completely migrated to the DRAM of the destination hybrid memory module. Also, access to the requested page is served from the destination DRAM only after the completion of the page migration. Therefore, regular application execution does not get obstructed due to page migration. Also, as the size of the DRAM at each hybrid memory module is small, therefore to write the page at the destination DRAM memory, page migration first considers the invalid row position of the DRAM memory at the destination hybrid memory slice. However, if all the rows are valid at the DRAM, then the page migration considers the row associated to the page having least frequently used page (LFU) for the writing, which is same as our considered regular DRAM to PCM page eviction method. Simultaneously, update the global page table.

TLB update mechanism: Following the page migrations, the global page table changes, so TLB of each core needs to be updated. In this chapter, there are two scenarios when TLB update need to be done and these are, (a) TLB update associated to the page movement from PCM to DRAM memory of a hybrid memory module, and (b) TLB update when page movement happens from a hybrid memory module to another hybrid memory module (which is similar to as explained in Chapter 5). There are many efficient TLB update algorithms available in the literature, and some are explained in Chapter 5. For simplicity, we have used the Lazy TLB update method (as explained in Chapter 5) for both the scenarios associated with the TLB update. Also, our work is not restricted to any TLB update mechanism.

Moreover, in this case, also, *to enable the run-time page mapping between the hybrid memory slices, we have assumed that the caches of the CMP system are addressed by the virtual address instead of the physical address.*

Further, page vp_j is marked as migrated by setting $Mig[vp_j]$ corresponding to the

TH-2297_136201001

TRUE values in the waiting queue WQ (refer Fig. 6.4) and all the entries associated to page vp_j is flushed out from the current hybrid memory module. The first phase onward, the values of the current epoch profiling counter C_{cntr} corresponding to the previous phase becomes the previous profile-counter P_{cntr} for the current phase. Moreover, at the end of first time epoch, we have considered zero values for the previous profiling counter P_{cntr}, so only current epoch profile-counter C_{cntr} values are used to calculate correlation $Cor[vp_j]$.

6.2 Access-Aware Page Placement Between DRAM and PCM of the Hybrid Memory Slice

Many researchers have explored 3D-stacked hybrid memory architect using PCM as well as DRAM as an alternative to only PCM or DRAM memory [126, 96, 95]. Typically, these hybrid memories regulate the placement of their data (or pages) to minimize the leakage power of the DRAM memory and the high access latency of the PCM memory. Therefore, considering that the leakage power due to the DRAM refresh is more prominent in the future DRAM and PCM based 3D-stacked hybrid memory, this section presents a simple access aware page placement between DRAM and PCM to avoid the DRAM refresh operation. To achieve this goal, we have used the retention time as well as access information related to each row of the DRAM, similar to as given by Pourshirazi *et al.* in [93].

Moreover, we performed row-level periodic monitoring of the DRAM rows (assuming a page of 4KB resides in a DRAM row) and based on the last access time we categorize the DRAM rows into *live row* and *decayed row*. A live row is the one that got recent access and therefore activated and refreshed. However, a decayed row has not been got access for a long time and thus very likely to carry inactive data. Therefore, to avoid power overhead related to the refresh, we can evict the decayed row from the DRAM to PCM at the end of the monitoring period (termed as *decision point*). Thus, all the refresh operations related to the decayed and inactive rows can be eliminated.

We have considered the DRAM row monitoring period equal to half of the retention time, which is similar to as considered by Pourshirazi *et al.* in [93]. In [68], Liu *et al.* have demonstrated that for 64nm 32GB DRAM, approximately 10^{11} number of cells have 256ms as their retention time. Whereas, about 30 and 1000 cells have

TH-2297_136201001

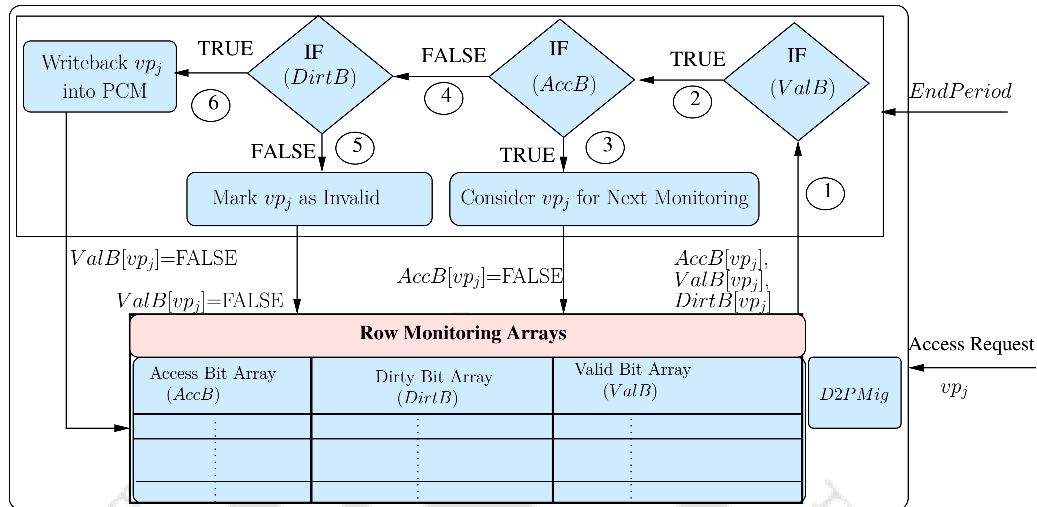


Figure 6.5: DRAM refresh controller architect inside the modified MC

the retention time value of 64ms and 128ms, respectively. Also, they have proposed a retention time aware DRAM refreshing technique. By assuming 1030 cells out of 10^{11} cells as inactive cells, for simplicity and to make the hardware cost low, we have considered an equal retention time value of 256ms for all the rows. Therefore, we have used the row monitoring period value as 128ms (which is equivalent to 3.44×10^8 cycles for our case) for the evaluation and considering the DRAM cells which are having less retention time can be discarded from the die.

Fig. 6.5, shows the detail structure of the **DRAM refresh controller** architect inside the modified memory controller (Fig. 6.4). For a hybrid memory module (or hybrid memory slice), we have considered that initially, all the pages remain in the PCM of the hybrid memory slice (or module).

As described in subsection 6.1.1, line number 5 to 13 of the Algorithm 6 shows the steps associated with the row monitoring performed by the DRAM refresh controller. Specifically, for each access request to a page vp_j and its DRAM row, the access bit array $AccB[vp_j]$, dirty bit array $DirtB[vp_j]$, and valid bit array $ValB[vp_j]$ status is maintained to keep track the access, write back, and validity (valid or invalid) information respectively. Each DRAM refresh controller performs the row monitoring operation periodically for a monitoring period of 128ms (or 3.44×10^8 cycles) by maintaining the status of $AccB[vp_j]$, $DirtB[vp_j]$, and $ValB[vp_j]$ arrays.

Moreover, at the end of a monitoring period, the *EndPeriod* pulse is triggered, and DRAM refresh controller starts the decision making about the page placement between the DRAM and PCM memory of a hybrid memory slice. Algorithm 8 shows

TH-2297_136201001

the steps to be followed at the decision point based on the row monitoring performed (as per line number 5 to 13 of the Algorithm 6). At the decision point, all the valid rows (having a valid page) of the DRAM are being checked one by one ①. Therefore, for a page in a valid row ②, if the associated access array value $AccB[vp_j]$ is "TRUE" then the row is considered for the re-monitoring by setting $AccB[vp_j]$ as "FALSE" and remains in the DRAM itself ③.

Whereas, for a page in the valid row, if the $AccB[vp_j]$ value is "FALSE" then the row is either get evicted from the DRAM or written back to the associated PCM based on the dirty bit array $DirtB[vp_j]$ value ④. If the $DirtB[vp_j]$ has "FALSE" value for the row (associated to the page vp_j) then the row is evicted from the DRAM and all entry associated to the row (for example dirty bit array, access bit array) is flushed from the DRAM ⑤. Also, "TRUE" value of the $DirtB[vp_j]$ for the row starts the writing back the page into the PCM ⑥. Also, the row monitoring array $ValB[vp_j]$ is updated after the completion of the writing into the PCM.

However, writing a page to the PCM need to read the page from the DRAM and also it takes time to write it in the PCM. We have considered a lazy approach for the DRAM to PCM page migration (or write). Also, using the fact that reading a row in the DRAM can be equivalent to the access request for the row and refreshes the page automatically, we have considered that until the page writing (or migration) to the PCM is finished future access to the page is allowed from the DRAM in parallel with the writing into the PCM. Once, writing into the PCM is completed, all the associated entry flushed out from the DRAM and page stays in PCM only.

6.3 Experimental Result and Analysis

In this work, we have used Sniper simulator [23] as the platform to configure the considered target chip multiprocessor system. We have used a 64-core (8×8 2D-mesh interconnection based) CMP system with four memory controllers (MCs). The hybrid memory modules of our system are considered as non-coherent and are not allowed to store multiple copies of the pages in between them. Moreover, our mapping technique is also applicable to larger CMP systems having an optimal number of memory controllers and their placements (can be decided using [90, 5]). Further, we have used Intel Xeon X550 Gainestown micro-architecture configuration to simulate each core of the CMP system, and other CMP configuration parameters are given

TH-2297_136201001

Algorithm 8 :Working of the DRAM Refresh Controller Inside Each Modified Memory Controller

```
1: if EndPeriod=TRUE then
2:   for All the valid DRAM rows (or pages) vp_j do
3:     if AccB[vp_j]= TRUE then
4:       Set AccB[vp_j] to FALSE.
5:       Consider the page for the next monitoring period.
6:     else
7:       if DirtB[vp_j] = FALSE then
8:         Set row associated with vp_j as invalid.
9:       else
10:        Write back the page vp_j into PCM.
11:        Set row associated with vp_j as invalid.
12:      end if
13:    end if
14:  end for
15: end if
```

in Table 6.1. Moreover, we have modeled each memory controller (MC), associated hardware and performance overheads inside the simulator, and the resulting latency is fed into the simulator. We have considered that DRAM refresh energy and PCM access energy is 5× and 10× respectively as compared to the DRAM access energy.

To perform the system performance evaluation, we have used multiple benchmark applications taken from the multi-threaded Cilk-5 [42] and SPLASH-2 [118] benchmark suits. An application that generates a higher number of last level cache misses (private L2 cache in our case) and having more substantial execution time is selected for the performance evaluation.

Fig. 6.6 shows the normalized performance (in terms of the execution time) for the different benchmarks. Also, Fig. 6.7 shows energy consumption in terms of DRAM access energy. In Fig. 6.6 and 6.7, "BC", "BCMap", "BCPCM", and "BCPCMMap" refers to the results corresponding to the following configuration.

- **BC**: In this configuration, the CMP memory layer consists of only DRAM memory (PCM memory is not considered) with an overall size of 32GB and 8GB per hybrid memory module (or slice). In this case, DRAM access-aware adaptive run-time page mapping is not used (base-case for our work).
- **BCMap**: In this configuration, the CMP memory layer consists of only DRAM memory (PCM memory is not considered) with an overall size of 32GB and 8GB

TH-2297_136201001

Table 6.1: System configuration parameters

Parameters	**Values**
Number of tiles	64
Number of cores per tile	1
Number of threads per core	1
L1-I and L1-D cache size	16KB per core
L2 cache size	128KB per core
Overall 3D-stacked DRAM	1GB, 256MB per MC
Overall 3D-stacked PCM	32GB, 8GB per MC
Cache block size	64 bytes
Memory Page size	4 KB
Coherence protocol at L2 Level	MESI
DRAM access latency	160 cycles
PCM access latency	640 cycles
Hop to hop latency	40 cycles
Value for Cor_{Thr}	100
Each Epoch (or Phase) size	10^9 cycles
Each Row Monitoring Period size	128ms (or 3.44×10^8 cycles)

per hybrid memory module. However, DRAM access-aware adaptive run-time page mapping is performed in this case. Also, as no PCM memory is considered, all the pages remain in the DRAM, and DRAM needs a periodic refresh to make the data integrity considering all the valid DRAM pages.

- **BCPCM**: In this configuration, we have considered that the hybrid memory architect using DRAM as well as PCM memory layers, and each hybrid memory module (or slice) associated with a memory controller is having 256MB DRAM memory and 8GB PCM memory. However, we have not used the DRAM access-aware adaptive run-time page mapping for this case.

- **BCPCMMap**: This configuration considers the benefit of the hybrid memory (architect using DRAM as well as PCM memory layers) as well as DRAM access-aware adaptive run-time page mapping. Also, we have considered a hybrid memory module (associated with a memory controller) as the combination of 256MB DRAM memory and 8GB PCM memory.

Fig. 6.6 shows the execution time comparison between the four configurations "BC", "BCMap", "BCPCM", and "BCPCMMap". The result shown in Fig. 6.6 has normalized to the base-case "BC" configuration. On an average, "BCMap" and

TH-2297_136201001

"BCPCMMap" configurations have reduced the overall execution time by an average of 4.27% and 4.23% respectively. Moreover, "BCMap" and "BCPCMMap" configurations have reduced the overall execution time up to a maximum of 11% for ocean.cont benchmark application. Reduction in "BCMap" configuration is due to the use of the page mapping that reduces the remote accesses, whereas this configuration is neutral while dealing with the DRAM row refresh overhead. However, technique "BCPCMMap" uses the benefits of the PCM component of the hybrid memory slice as well as the page mapping. Also, for the CMP having larger core count, the remote access latency (due to the increased hop count in the interconnection network) dominates the memory access latency; therefore, execution time in "BCPCM" configuration is almost same to the base-case "BC".

Fig. 6.7 shows the normalized energy consumption at the 3D-stacked memory layer for the different considered configurations "BC", "BCMap", "BCPCM", and "BCPCMMap". In this figure, we can see that method "BCPCMMap" consumes on an average of 49% less energy as compared to the "BC". This reduction in energy is mainly due to the following reasons: First, configuration "BCPCMMap" takes the benefits of the DRAM access aware page placement between DRAM and PCM memory to reduce the DRAM refresh operation. Second, the access-aware self-adaptive page mapping takes the benefits of the page access (while migrating a page from a hybrid memory slice to other) mechanism and refreshes the DRAM row. Therefore, it reduces the DRAM refresh energy by avoiding the refresh operation for those migrated pages.

Therefore, together from Fig. 6.6 and 6.7, we can see that "BCPCMMap" gives the similar or better performance (in terms of the execution time) as compared to the "BC". Moreover, "BCPCMMap" configuration also reduces the energy consumption by lowering the DRAM refreshes.

6.4 Performance and Area Overhead Analysis

6.4.1 Performance Analysis

In this work, we have considered the page size of 4KB for the DRAM memory as well as PCM memory of the hybrid memory slice. Both parts, (a) access-aware self-adaptive page mapping on to the hybrid memory slices, as well as (b) DRAM access-aware page placement between the DRAM and PCM memory of the hybrid

TH-2297_136201001

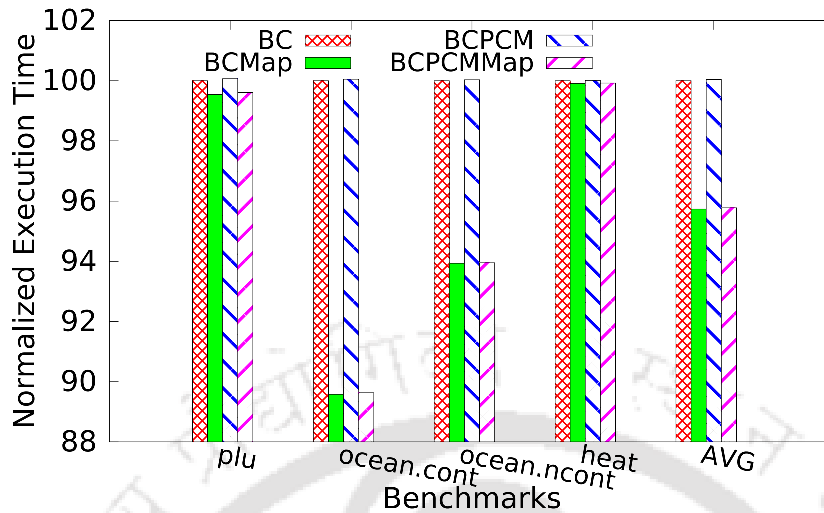


Figure 6.6: Normalized execution time

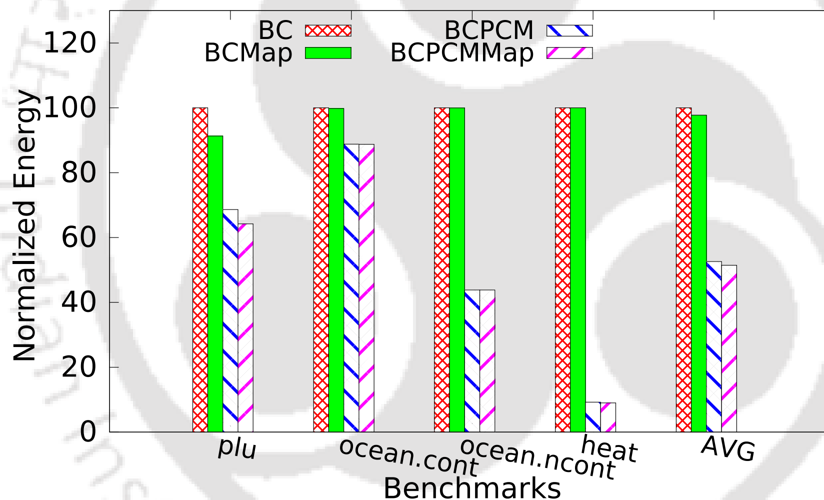


Figure 6.7: Normalized energy consumption considering the DRAM refresh energy and PCM access energy

memory slices, of our approach generates the page migrations. Former one generates the pages migrations in between two hybrid memory slices and termed as inter-slice page migrations. Whereas, latter one causes the page migrations between the DRAM and PCM of a hybrid memory slice and termed as intra-slice page migrations.

Similar to the technique as described in Chapter 5, this chapter also uses the lazy page migration technique for the inter-slice page migration, so there is no obstruction in the regular application execution time. To analyze the percentage of overall page migrations, in this chapter also we performed the experiments similar to as done in Chapter 5. Our experiment shows that the number of the page migration due to the

TH-2297_136201001

run-time self-adaptive page mapping is similar to the number of page migration, as shown in Fig. 5.3 of Chapter 5.

For the intra-slice page migration, pages that are not dirty can be invalidated merely by setting $valB$ array. Whereas, we need to migrate (or write) the pages from DRAM to PCM of a hybrid memory slice only in case of write-back (or being dirty in DRAM). We have used a one-bit array $D2PMig$ that set to "TRUE" if a DRAM row (associated to a page) becomes the candidate for the DRAM to PCM migration. Further, we have used a simple distributed lazy method to perform the DRAM to PCM page migrations decided at the **decision point** after each row monitoring period. Also, in this method, a higher priority is given to the regular DRAM to PCM read access (generated from the DRAM memory) as compared to the page migrations.

In the distributed lazy page migration, to perform a page migration, it is written to the PCM at the smaller granularity and spread over the row monitoring period instead of writing a page entirely at a time. Therefore, to migrate a 4KB page, we have considered writing 64bytes at a time, and a complete page is written using 64 smaller chunks of 64bytes. Also, after writing a block of 64byte and before starting the next write, it checks for the regular DRAM to PCM read access and preference is given to the PCM read if it is present. Moreover, once a page migration (or PCM write) is get completed, associated $D2PMig$ array is set to "FALSE" along with the updation of the associated row monitoring arrays $AccB$, $ValB$, and $DirtB$. Similarly, the next candidate page is considered for the migration. However, if the number of pages to migrate between DRAM and PCM became high, then the pages that could not migrate during the row monitoring period will get considered for the upcoming decision point (as the DRAM row can retain till 256ms).

6.4.2 Area Overhead Analysis

As we have considered, 4KB page size and 4 hybrid memory slices (M=4) with the DRAM memory size per hybrid memory slice D_{size}= 256MB, so there are $4\times\frac{256MB}{4KB}$ number of physical pages that can reside on the DRAM memory of the DRAM-PCM based hybrid memory. Also, we need counters for the pages that are in the DRAM memory and not on the PCM memory while performing the profiling, and therefore, for that, we need $4 \times \frac{256MB}{4KB} \times 2 \times 4$ the number of 16-bit profiling counters.

Further, to conduct the computation while deciding the page mapping, it incurs $(M+1)$ number of the comparator, and also M number of adders. Moreover, for each

TH-2297_136201001

physical page, it takes $log_2(M)$ cycles to perform the computations for the mapping using the M number of the adder and $M+1$ number of comparator units. Therefore, for $M = 4$ hybrid memory slices, the modified memory controller incurs a hardware overhead of 4 adders and comparator along with the time overhead of 4 cycles. Also, to create the migration list Mig and waiting queue WQ, we need an overall of $4 \times \frac{256MB}{4KB}$ bit or 32KB of memory space for each. Similarly, location array Loc for the destination hybrid memory module incurs $4 \times \frac{256MB}{4KB}$ 2-bit or 64KB memory space. Moreover, for profiling counters, we need 4MB memory space corresponding to 1GB of total DRAM memory.

Moreover, to perform the DRAM to PCM page placement considering the overall hybrid memory, we need 32KB memory corresponding to the $AccB$, $valB$, $DirtB$, and $D2MMig$ one-bit arrays at each DRAM refresh controller. Also, we need an overall of $3 \times M$ one-bit comparator.

6.5 Summary

In this chapter, we have proposed an access-aware self-adaptive run-time page mapping for the 3D-stacked hybrid DRAM-PCM memory-based CMP system. Our proposed method uses a simple DRAM access-aware page placement technique between DRAM and PCM of the hybrid memory to reduce the DRAM refresh operations and its associated power consumption overhead. Further, it uses the DRAM row access information and performs an access-aware self-adaptive page mapping for the optimized page placement between the different hybrid memory modules of the 3D-stacked hybrid memory. In this way, our method provides an alternative and efficient way to use 3D-stacked hybrid DRAM-PCM memory for the future generation CMP system.

We expected our technique to become more effective in the future large CMP system (core count ≥ 100) having larger on-chip memory (to fulfill the high memory bandwidth demand). Our proposed approach performs similar or better in terms of the execution time as compared to the base case. Our technique reduces the energy consumption due to the DRAM refresh by an average of 51% as compared to the base case. Moreover, our method incurs a small area overhead in the memory controller at the 3D-stacked memory layer.

TH-2297_136201001

7

Performance Analysis of CMP having 3D-stacked DRAM and Hybrid NOC

The advancement in the fabrication technology of the CMP system has the potential to increase the core count very significantly. However, the benefits of the increasing number of cores on a CMP system are limited by many architectural constraints and resource management techniques. The off-chip memory bandwidth can severely restrict the core count of the CMPs and thereby reduces the performance, as reported by Rogers *et al.* in [100]. Also, the increase in the CMP core count is urging the necessity of not only the high bandwidth memory but also the high-speed on-chip interconnects or network-on-chip (NOC).

As described in the previous Chapters, 3D fabrication technology becomes popular, as it allows us to stack DRAM memory and non-volatile memories on top of the chip to addresses the system bandwidth and performance demand [20, 126, 85]. Recently, researchers have started exploring many on-chip interconnect technologies, including wireless NOCs and 3D-stacked optical NOCs [35, 123, 61, 112]. For the CMPs, optical NOC has been explored extensively in the last few years due to its easy implementation using the 3D fabrication technology [123, 61, 113]. In 3D fabrication, all the optical components associated with the optical NOC are placed at a separate optical layer, and these are interfaced with the electrical components using TSVs [113]. Optical on-chip interconnect either supplement the electrical intercon-

TH-2297_136201001

nection network or replaces it entirely. In [114], Werner *et al.* have proposed a hybrid interconnect technique for the CMP systems. Moreover, they have identified that the electrical links are best suited for the near distance communications and optical links perform well when the destination is in the two-hop distance or more.

Agarwal *et al.* in [7], studied that for a fixed die size, we need to reduce the cache size per core to increase the core count of the CMP. Therefore, the trade-off between performance and cache size per core need to be analyzed along with the use of 3D-stacked memories as well as a high-end interconnection network. Also, conclusions from the last chapters made it clear that the self-adaptive run-time page mapping is an effective way to use the 3D-stacked memories for the current as well as future CMP systems.

Therefore, in this chapter, our objective is to study the trade-off between the performance and cache size per core of the CMP system while utilizing the benefits of the optical interconnect, 3D-stacked DRAM and a self-adaptive run-time page mapping. Specifically, we analyze the effects of reduced cache size on to the system performance while considering the benefits of the following.

1. 3D-stacked DRAM, to provide high memory bandwidth.
2. High performance optical interconnect, to enable low-latency communication and efficient memory utilization.
3. Finally, we use the self-adaptive run-time page mapping to efficiently use the 3D-stacked DRAM for the CMP system, as proposed in chapter 5.

7.1 Target System Architecture

In this chapter, we have considered a 3D-stacked DRAM based CMP system having an optical interconnection network, as our target CMP system. Figure 7.1 shows an example of the considered 3D-stacked CMP architecture, in addition to the detailed representation and explanation as given in Section 3.1.4 and Chapter 3. In Figure 7.1, processor layer has 64 cores that are connected using an 8×8 two-dimensional mesh of electrical interconnects. In addition to the 2D-mesh electrical interconnects, there is an optical network present above the processor layer, and together they form a hybrid interconnection network for the CMP system. Optical interconnect layer on top of the processor layer is having four optical network interfaces (oni_0, oni_1, oni_2,

TH-2297_136201001

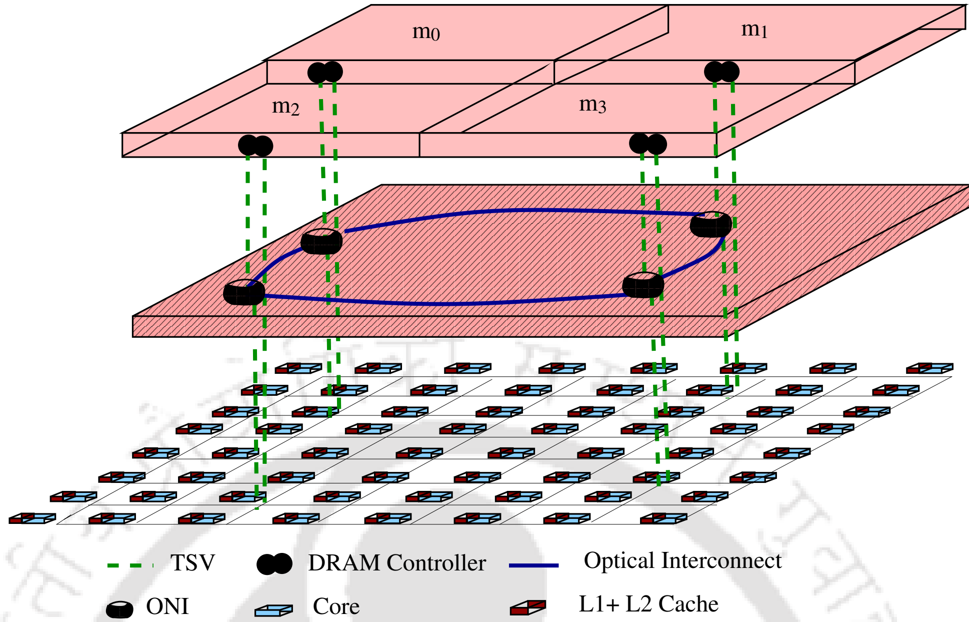


Figure 7.1: CMP system architecture.

$oni_3 \in ONI$) and these are connected in a ring through the optical interconnects (waveguide). The high bandwidth optical network is shown by a thick line in Fig .7.1. Further, a memory layer is placed on top of the optical interconnect layer. Memory layer comprises four $\{m_0$, m_1, m_2 and $m_3\}$ 3D-stacked DRAM memory slices (or memory banks) and each memory slice is having their own on-chip memory controller.

7.1.1 Routing of Packets

We have used modified version of the XY-routing to route the packet in the considered hybrid interconnection network. Considering the $CMHIG$ graph as shown by Figure 7.2 (right part of Figure 3.6, as explained in section 3.1.4) corresponding to the target CMP system, for the communication between any pair of source NOC-tile vertex c_s and destination NOC-tile vertex c_d. The routing between c_s and c_d is decided based on the fact that: “if the use of optical network is beneficial as compared to the only electrical interconnect then former one should be preferred for the path selection and packet routes through a pair of intermediate nodes”. Suppose, c_x and c_y are the nearest NOC-tile vertices (termed as intermediate nodes and are having adjacent ONI) from the NOC-tile vertices c_s and c_d respectively. Consider, the electrical distances (in number of hops$\times$ electrical link latency per hop) (a) from c_s to c_d as D_{Esd}, (b)

TH-2297_136201001

from c_s to c_x as D_{Esx} and, (c) from c_y to c_d as D_{Eyd}. Also, the optical distance (in number of hops$\times$ optical link latency per hop) between between intermediate nodes c_x and c_y as D_{Oxy}. The routing of the packets from c_s to c_d uses the optical path if $D_{Esd} > (D_{Esx}+D_{Oxy}+D_{Eyd})$ becomes true. For the true value of this condition, the electrical path get split into, (1) electrical path from c_s to c_x, (2) electrical path from c_y to c_d, and (3) optical path from c_x to c_y.

At every processor layer router, the information about the intermediate node is specified and being used for the optimized routing. Moreover, routing between each ONI at the optical interconnect layer as well as each router at the processor layer takes place using dynamic XY-routing based on the input queue traffic. In the dynamic XY-routing, to avoid the congestion, next hop is decided based on the current queue length of the corresponding input port in the neighboring routers [65]. Therefore, for a packet that needs to travel some distance in both X and Y directions to reach the destination, dynamic XY-routing routes the packet to either X or Y direction depending on the congestion conditions of the input queues associated to the neighboring routers. Also, the input port of the neighboring routers having a lower queue length is preferred. For an example: a packet from NOC-tile vertex c_2 to the c_{61} use the route through intermediate node c_{17} and c_{46}. However, the packet traversal between c_2 to c_{17}, c_{17} to c_{46} and c_{46} to c_{61} uses dynamic XY-routing algorithm.

The intermediate nodes to be used for any pair of nodes (NOC-tile vertices) are computed statically and stored once at every router of the NOC-tile vertex to route the packets. A table (termed as VIA table) at every router of the NOC-tile vertex specifies the intermediate nodes which are used to go to other NOC-tile vertices. VIA table at each router keeps the information for all the destinations with their associated intermediate nodes. For example, Table 7.1 shows a segment of the VIA table stored at the router of the NOC-tile vertex c_0 for each destination and values of the associated intermediate node NOC-tile vertices to go through. Value -1 is used when there is no specified intermediate node to be used for a source and destination NOC-tile vertex pair.

For 8×8 CMP system architecture having four ONIs, we need to compute and store the routing table at each router. As the number of ONIs are four and need to represent the -1 (NULL) value, therefore it incurs 3-bits of memory to represent each intermediate node in VIA table. Overall, it incurs an overhead of $64\times$(6 bit

TH-2297_136201001

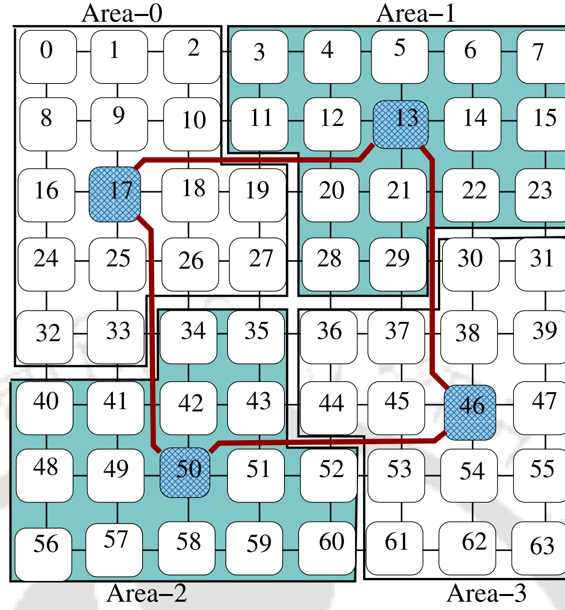


Figure 7.2: Example of $CMHIG$ graph

Source	Destination	$\mathbf{IN_{src}}$	$\mathbf{IN_{dst}}$
c_0	c_0	-1	-1
c_0	c_1	-1	-1
c_0	c_2	-1	-1
.	.	.	.
c_0	c_7	c_{17}	c_{13}
.	.	.	.
c_0	c_{63}	c_{17}	c_{46}

Table 7.1: A segment of the VIA table stored at router of the NOC-tile vertex c_0, where: $\mathbf{IN_{src}}$ and $\mathbf{IN_{dst}}$ are intermediate nodes adjacent to source and destination nodes respectively.

for destination+3 bit for first intermediate node+3 bit for second intermediate node) bits at each router.

7.2 Problem Formulation

Modern chip-multiprocessors (CMPs) dedicate 40-60% chip area to the caches. As the cache dominates in a CMP, increasing the size of the cache reduces the number of cores significantly for a fixed die size. For the present-day CMPs, the overall chip

TH-2297_136201001

area A_{Chip} can be given by (7.1).

$$A_{Chip} = N_c \times (A_{Core_{L1}} + A_{PrivL2}), \tag{7.1}$$

where, N_c is the number of cores on the chip, $A_{Core_{L1}}$ is the area of one core with L1 cache (data and instruction), and A_{PrivL2} is the area of the overall private L2 cache per core.

Table 7.2 shows the number of cores that can be accommodated on a CMP for the overall die size of $240mm^2$, and different combinations of the private L2 cache size and core size. For big size cores ($2.4mm^2$) and L2 cache size of 512KB per core, $240mm^2$ CMP die can accommodate 61 cores. Whereas, for 32KB per core L2 cache size, 96 big cores can be accommodated. Similarly, for tiny cores ($0.6mm^2$), 512KB per core L2 cache size results in 114 cores to be accommodated on CMP. Whereas, 32KB per core L2 cache size results in 346 tiny cores to fit on CMP, which is significantly higher. A larger number of cores per CMP results in higher performance if an adequate amount of memory feeds the cores and interconnect bandwidth.

An adequate amount of memory bandwidth can be provided by increasing the cache size, which is contradictory to increasing the number of cores for the fixed die size. Therefore, increasing the main memory bandwidth and reducing the delay between memory and cores by using high end optical interconnects can be an efficient alternative. In our considered architecture, cores and L2 caches reside at the processor layer; therefore, reduction in the L2 cache size per core increases the number of cores. 3D-stacked on-chip memory layer increases the memory bandwidth by stacking the DRAM slices on top of the processor layer. Further, efficient run-time data page mapping places the critical data to the nearby memory slice of the requesting core. The use of optical interconnects further reduces the remote memory access delay and thread to thread communication delay. Also, it reduces the network congestion overhead due to the page migration incurred after the page mapping by providing a separate path between the DRAM slices.

In this chapter, our goal is to analyze the effects of the reduced cache size per core on the CMP system performance by considering **(a)** the optical interconnects, **(b)** different size of the L2 cache per core and, **(c)** the use of self-adaptive run-time page mapping onto the DRAM memory slices.

Consider, the multi-threaded application representation $AGVP(\mathbf{T}, \mathbf{VP}, \mathbf{E_{tc}}, \mathbf{E_{vpa}})$, as given in section 3.2; and a target CMP representation $CMHIG(\mathbf{C_{cc}}, \mathbf{E_{Ecc}}, \mathbf{E_{Occ}})$, as

TH-2297_136201001

Core type & area mm^2	512KB $1.5mm^2$	256KB $0.75mm^2$	128KB $0.38mm^2$	64KB $0.19mm^2$	32KB $0.1mm^2$
Big (2.4mm^2**)**	61	76	86	92	96
Medium(1.5mm^2**)**	80	106	128	142	150
Small(0.8mm^2**)**	104	154	204	243	268
Tiny(0.6mm^2**)**	114	177	246	304	346

Table 7.2: Number of cores that can fit with different cache and core size for the CMP die area of $240mm^2$, using (7.1).

given in section 3.1.4. Our self-adaptive run-time mapping approach maps the multi-threaded applications represented as graph $AGVP$ onto target architecture graph $CMHIG$ to maximize the performance. We use default thread to core (associated to the NOC-tile vertex) mapping in this part of work. Moreover, in our mapping approach, we use page mapping technique considering the amount of thread to virtual page access edge $e_{vpa}(t_i, vp_j)$ and ignored the thread to thread communication edges $e_{tc}(t_i, t_j)$ to maximize the performance.

The miss latency $T_{L2M,t}$ incurred for thread t_i due to the last level cache (LLC) miss of a cache block (associated to a page vp_j) at time instant t, can be calculated as follows.

$$T_{L2M,t} = distCC(X(t_i), Y_t(vp_j)).L_{H2H} + L_{DRAM}. \tag{7.2}$$

Where $X(t_i)$ is the NOC-tile vertex associated with the core having mapped thread t_i. This $X(t_i)$ is the default thread to core mapping at the starting of the application execution and remains the same over the application execution. $Y_t(vp_j)$ is the NOC-tile vertex adjacent to the DRAM memory slice having mapped page vp_j at time instant t. The term $distCC(X(t_i), Y_t(vp_j))$ is the hop distance (or hop count, calculated using the routing approach as described in Sub-section 7.1.1) between NOC-tile vertex associated to $X(t_i)$ and $Y_t(vp_j)$. The term L_{DRAM} is the DRAM access latency to access the cache block of a page. L_{H2H} is the hop to hop traversal latency, which depends based on the interconnect type while calculating the hop distance using the routing method described in Section 7.1.1 using the hybrid network.

As most of the applications exhibit phase-wise behavior during their run-time [105]. Therefore, for an application, the LLC miss pattern of an execution phase may be different from another phase. So, we considered the run-time phase-wise behavior of an application and for simplicity divided the whole execution time of an application

TH-2297_136201001

into multiple time epoch (or phase) of fixed length. Therefore, using (7.2), the total latency overhead $T_{L2M,epo,i}$ incurred due to all the LLC misses from thread t_i (mapped to core c_i) in a time epoch *epo* can be given by (7.3).

$$T_{L2M,epo,i} = \sum_{j=0}^{|\mathbf{VP}|-1} \omega_{epo}(e_{vpa}(t_i, vp_j)).T_{L2M,t}. \tag{7.3}$$

Where, $\omega_{epo}(e_{vpa}(t_i, vp_j))$ is the number of page access request by thread t_i to virtual page vp_j in a time epoch *epo*. These page access requests of thread t_i is the LLC miss of processor c_i as t_i is mapped to c_i. The term **VP** represents the set of virtual pages and includes total number of pages corresponding to the multi-threaded application.

For E number of phases (or epochs) associated to the total execution time of an application, the overall latency $T_{L2M,overall,i}$ associated with the LLC miss from the thread t_i (mapped to core c_i) is given as follows.

$$T_{L2M,overall,i} = \sum_{epo=1}^{E} T_{L2M,epo,i}. \tag{7.4}$$

Therefore, considering the phase-wise behavior and parallel execution of the multi-threaded applications on to the CMP, the total time after the last level cache and due to the memory accesses T_{Total} of the application can be given as following (as explained by Equation 5.4 in chapter 5).

$$T_{L2M,total} = Max(T_{L2M,OverAll,i}, \forall i \in \{0, 1, 2, \ldots, N-1\}). \tag{7.5}$$

Where, $T_{L2M,OverAll,i}$ is the overall miss latency associated with the last level cache miss for i^{th} core c_i (having mapped thread t_i) over the E number of phases (or epochs) of the complete application execution.

Therefore, for the total number of instruction N_{ins} of the multi-threaded application and total execution cycle T_{Total} (in cycles), the instruction per cycle IPC of

TH-2297_136201001

the system is given by (7.6).

$$IPC = \frac{N_{ins}}{T_{Total}}. \tag{7.6}$$

Considering, the total execution cycle T_{Total} as sum of the total number of cycles $T_{L2M,total}$ (incurred after L2 cache misses) and the total number of cycles to perform computations T_{Cal}[1], IPC can be given by (7.7).

$$IPC = \frac{N_{ins}}{T_{L2M,total} + T_{Cal}}. \tag{7.7}$$

In the multiprocessor environment, where multiple instruction execution and memory page access happens in parallel, the serial calculation of delay $T_{L2M,total}$ is a reasonable estimate of delay value in calculating the performance. Equation (7.7), shows that if we reduce the value of $T_{L2M,total}$ then the value of IPC increases. The value of $T_{L2M,total}$ can be reduced by efficient placement of the memory pages on to the DRAM slices and low latency interconnection network.

Moreover, on-chip communication cost due to the memory page access can be evaluated by (7.8).

$$C_{Cost} = \sum_{epo=1}^{E} \sum_{i=0}^{N-1} \sum_{j=0}^{|\mathbf{VP}|-1} \omega_{epo}(e_{vpa}(t_i, vp_j)).distCC(X(t_i), Y_t(vp_j)). \tag{7.8}$$

Where, term $distCC(X(t_i), Y_t(vp_j))$ is the hop distance or hop count [2] between NOC-tile vertex associated to $X(t_i)$ and $Y_t(vp_j)$. $X(t_i)$ is the NOC-tile vertex associated to the core having mapped thread t_i. Term $X(t_i)$ is the default thread to core mapping at the starting of the application execution and remains same over the application execution. $Y_t(vp_j)$ is the NOC-tile vertex adjacent to the DRAM memory slice having mapped page vp_j at time instant t.

Equation 7.8 and 7.2, shows that decreasing the hop count (or hop distance) $distCC(X(t_i), Y_t(vp_j))$ proportionally reduces the on-chip communication cost as

[1] T_{Cal} includes computational cycles (at core) and total cycles to handle L1 hit/miss and L2 hit, used for the IPC calculation as given in [24]

[2] Calculated using the routing approach as described in sub-section 7.1.1 and it is similar to $distCC(X(t_i), Y_t(vp_j))$ as given in previous chapters, except the routing schemes used. In addition, we have assumed 1 hop distance between any two adjacent ONIs while calculating the communication cost and $distCC(X(t_i), Y_t(vp_j))$

TH-2297_136201001

well as miss latency of the CMP system. Therefore, for the CMPs having larger core count, efficient mapping and hybrid interconnect can reduce $distCC(X(t_i), Y_t(vp_j))$ significantly which in turn reduce the on-chip communication cost and total execution time. Reduced value of the execution time increases the system IPC as per the Equation 7.7. Moreover, for a fixed core size, if we reduce the last level cache (L2 cache in our cache) size then value of $\omega_{epo}(e_{vpa}(t_i,vp_j))$ increases (due to the higher number of miss generated) and so on-chip communication cost and miss latency value increases.

Therefore, for modern as well as future CMP system, it becomes essential to analyze and find an architectural design such that it maximizes the IPC and minimize the on-chip communication cost. So, in this contribution, our main aim is to analyze the effects of the different cache size per core on CMP system IPC and on-chip communication cost C_{Cost}. Also, if the performance of the CMP system with smaller cache size per core increases then we can utilize the chip area to increase the number of cores, and this may further increase the performance for the highly parallel applications.

7.3 Self-adaptive Application Mapping

In this chapter, we have considered the self-adaptive run-time page mapping as explained in Chapter 5, to effectively utilize the 3D-stacked DRAM memory while analyzing the effects of the reduced cache size on to the system performance. Therefore, for every last level cache miss generated to a cache block (associated to a page vp_j mapped to the DRAM slice $m_k \in \{m_0, m_1, \cdots, m_{M-1}\}$) from a core (having mapped thread $t_i \in \mathbf{T}$), associated core sends cache block access request to the memory controller of the DRAM slice having associated virtual page vp_j. Moreover, for each such access request (denoted by pair (t_i, vp_j, pp_j)), self-adaptive run-time page mapping performs its associated page access, run-time profiling, page mapping, migration and TLB update mechanisms similar to the ways as described in previous chapters.

The only difference that self-adaptive run-time page mapping poses in this chapter is the use of optical interconnects and the modified routing technique (as mentioned in Section 7.1). Therefore, whenever a message needs to travel through the on-chip interconnection network, it follows the modified routing as described in Section 7.1.1 and based on this routing, it decides the optimal path to travel.

TH-2297_136201001

7.4 Experimental Environment

In this contribution, we have used the Sniper simulator [23] platform to configure the considered target chip multiprocessor architecture. The architecture configuration used is 64 cores (8×8) CMP system with four DRAM controllers and four optical interconnects. Intel Xeon X550 Gainestown micro-architecture along with the private L1-I and L1-D of 8 KB and different sizes (32, 64, 128, 256, and 512 KB) of L2 cache per core is used to configure each simulated core of the target CMP system. Also, Table 7.3 shows the other configuration parameters used in this work. For our optical interconnect model, we have assumed the optical link latency of 4 cycles between two ONIs (including the delay of all the optical components). Moreover, we modeled the DRAM controllers and ONIs, associated hardware and performance overheads and fed the resulting latencies to the simulator for correct timing simulation.

For evaluation, we have used workloads from the multi-threaded PARSEC [14] and SPLASH-2 [118] benchmark suits. Workloads from the PARSEC [14] and SPLASH-2 [118] benchmark suits which generate a higher number of the last level cache misses (private L2 cache in our case) are used for the performance evaluation. We run the multi-threaded workloads with *simlarge* input for 10^8 instruction count.

In this work, we have assumed that queuing delay has less effects on system performance as compared to the latency delay due to the interconnects (on-chip and off-chip). Moreover, the same is supported by the summary Table 7.4 for the used benchmarks, which shows that all the used benchmarks have low miss per kilo instruction (MPKI) at the last level cache, which gets translated to low injection rate at the routers of the CMP system. Therefore, we did not model the "queuing delay" for the optical interconnects at the junction points of the electrical and optical interconnects. Furthermore, the used Sniper simulator considers and models the effects of the "queuing delay" at the electrical 2D-mesh network. Also, our results presented in this work are produced using the simulator, which considers and models the effects of the queuing delay for the electrical 2D-mesh network.

7.5 Results

The 3D-stacked DRAM in our considered CMP does not allow the storage of multiple copies of the virtual pages for the remote accesses and hence avoids the overhead associated with the coherence maintenance. Run-time self-adaptive virtual page to

TH-2297_136201001

Table 7.3: System configuration parameters.

Parameters	**Values**
Number of tiles	64
Number of cores per tile	1
Core frequency	2.6GHz
Number of threads per core	1
Overall 3D-stacked DRAM	4GB, 1GB per DC
DRAM directory type	Full map
Electrical link bandwidth	16 (bits/cycle)
Electrical link latency	10 (cycle/per hop)
Optical link bandwidth	256 (bits/cycle)
Optical link latency	4 cycles maximum
3D-stacked DRAM access latency *(including negligible TSV traversal delay)*	40 cycles
DRAM frequency (for on-chip and off-chip both)	800MHz
Size of Processor load-store Queue	8
Cache replacement policy	LRU
DRAM directory type	Full map
Off-chip DRAM access latency *(including long off-chip wire traversal delay, and used only for the "BC")*	400 cycles
Network Contention considered	No
Value for Cor_{Thr}	100

Table 7.4: Summary of the benchmarks characteristics

Benchmarks	**Application Domain**	**Average MPKI at L2 Cache**			
		64KB	128KB	256KB	512KB
ferret	Similarity search	7.07	6.08	4.94	4.09
x264	Media processing	18.89	11.78	3.96	2.72
barnes	N-body method	3.55	1.14	0.69	0.54
fmm	Fast multipole method	1.17	0.83	0.73	0.60

memory slice mapping improves the performance by placing the highly demanded remote virtual pages of a core to its nearest DRAM memory slice. Further, the use of efficient optical interconnects and routing technique reduces the on-chip distance significantly between two distant NOC-tiles (or core vertices), which in turn optimizes the communication cost and access latency. Also, network traffic generated due to the page migration between the DRAM slices is tackled by the optical interconnects as all the page migrations use the optical interconnects only. Whereas, based on the routing mechanism, other message packets may also route through the optical

TH-2297_136201001

interconnects.

To perform our proposed trade-off analysis, we have used different architectural cases of the interconnection network and placement of the DRAM memory (off-chip and 3D-stacked on-chip) along with the self-adaptive run-time page mapping. These cases are given as follows.

1. **BC:** (base-case for this work) A CMP system with off-chip DRAM memory and without a hybrid network is considered for this case. The run-time self-adaptive page mapping is not used for this case. We have used the one-hop electrical distance between off-chip DRAM slices and processor layer specific router. Also, for this case, we have considered 400 cycles as the access latency of the off-chip DRAM memory.

2. **STKDRAM:** In this case, a CMP system with 3D-stacked on-chip DRAM memory and without a hybrid network is considered. The run-time self-adaptive page mapping is not used in this case.

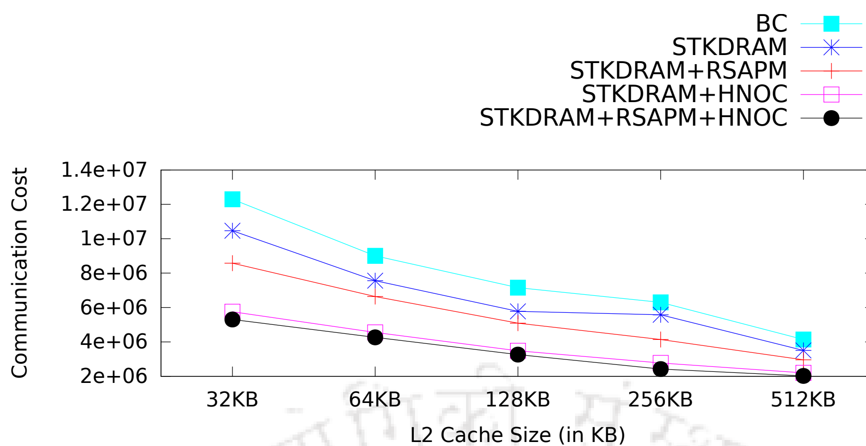
3. **STKDRAM+RSAPM:** In this case, we have considered a CMP system with the 3D-stacked on-chip DRAM memory and without hybrid interconnection network. We have used the run-time self-adaptive page mapping in this case.

4. **STKDRAM+HNOC:** A CMP system with the 3D-stacked on-chip DRAM memory along with the hybrid interconnection network is considered for this case. In this case, we have not used the self-adaptive run-time page mapping.

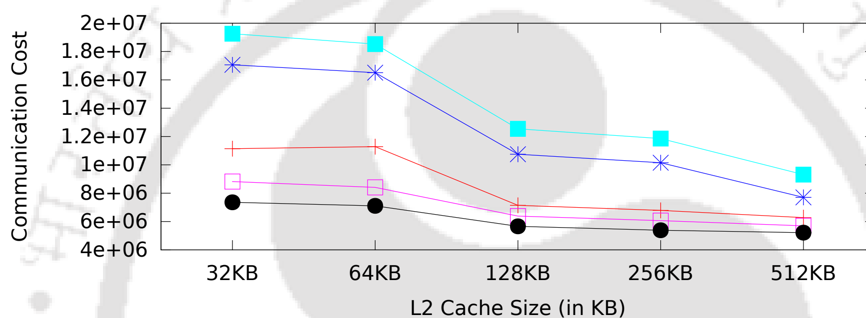
5. **STKDRAM+RSAPM+HNOC:** In this case, a CMP system with the 3D-stacked on-chip DRAM memory along with the use of the hybrid network is considered. Also, we have used the run-time self-adaptive page mapping for this case.

Figure 7.3 shows the overall communication cost (in the number of memory access $\times$ hop count) associated with the barnes, ferret, x264 and fmm benchmarks for the different values of the L2 cache size per core to estimate the power consumption of the network-on-chip (NOC). Similarly, Figure 7.4 shows the instruction-per-cycle (IPC) associated with the barnes, ferret, x264, and fmm benchmarks. We have considered, different combinations of the interconnection network (only electrical or hybrid), page mapping technique, and placement of the DRAM memory (3D-stacked based on-chip or off-chip) for the result evaluation.

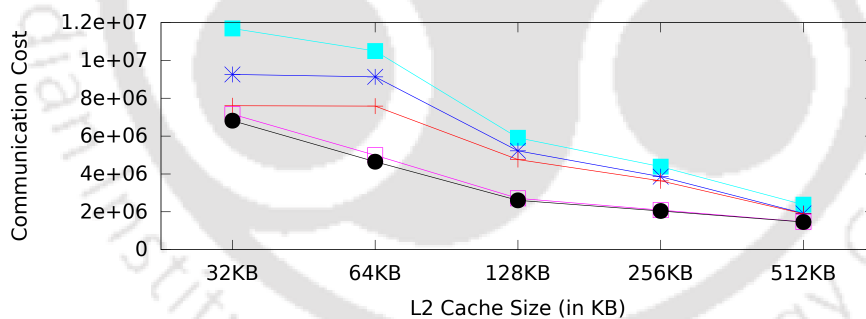
TH-2297_136201001



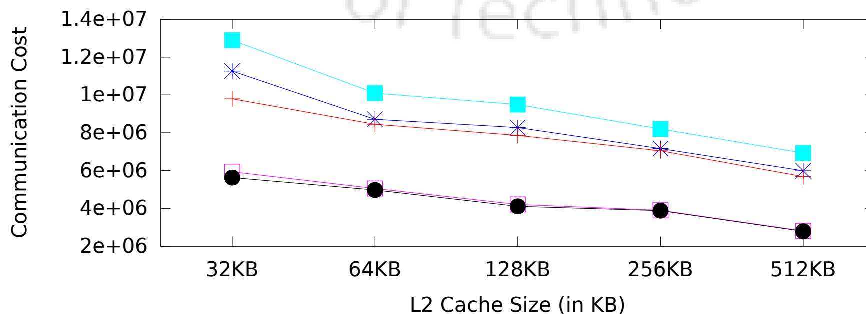
(a) barnes



(b) ferret



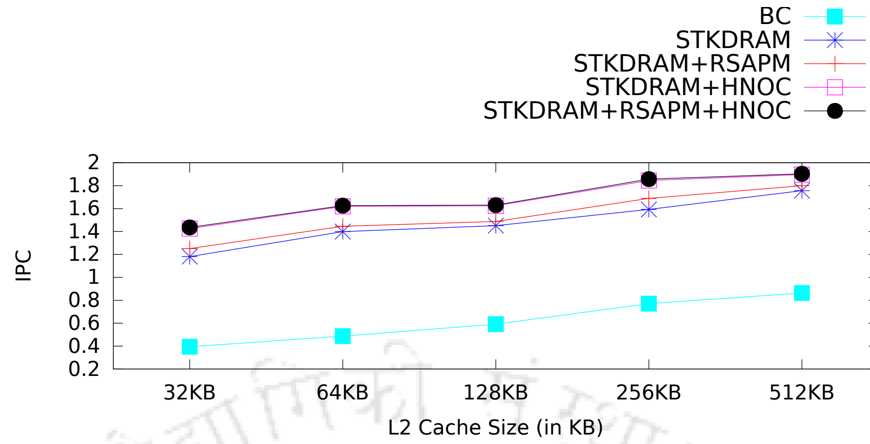
(c) x264



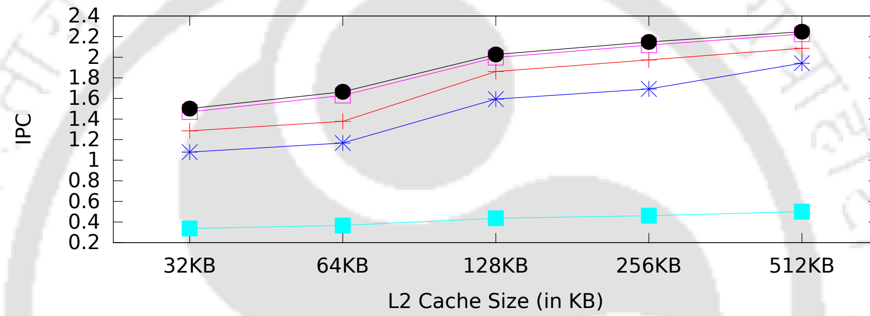
(d) fmm

Figure 7.3: On-chip communication cost (in number of memory access×hop cont) of the CMP system.

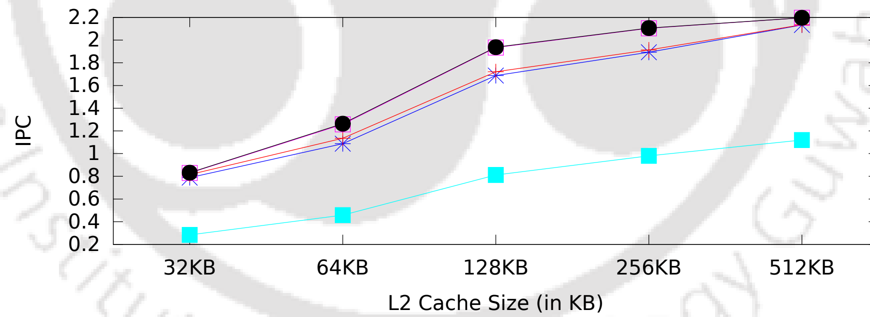
TH-2297_136201001



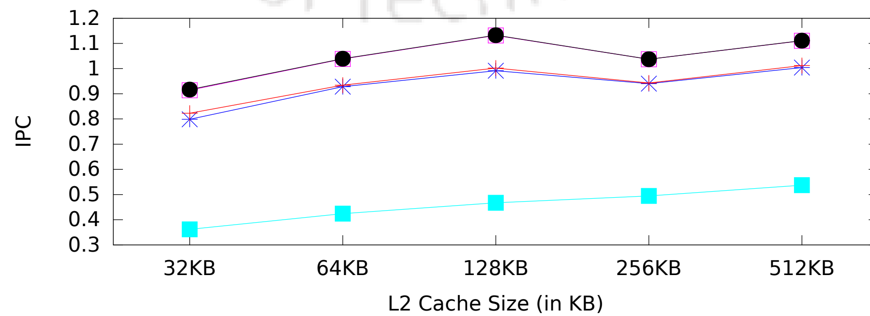
(a) barnes



(b) ferret



(c) x264



(d) fmm

Figure 7.4: Instruction per cycle (IPC) of the CMP system

TH-2297_136201001

Given a fixed die size, a CMP can have a higher number of cores if the size of the L2 cache per core is reduced from a larger value to a lower value. For example, considering a fixed die size of $240mm^2$, if the size of the L2 cache per core is reduced from 256KB to 128KB then as per Table 7.2, a CMP can have 10, 26, 50 and, 69 additional cores for big, medium, small and tiny size cores respectively.

Figure 7.3(a), 7.3(b), 7.3(c) and 7.3(d) shows that while increasing the L2 cache size per core the communication cost value of “BC” reduces. Also, if the size of the L2 cache per core is reduced from a larger value to a smaller one, then the communication cost value of the “BC” increases. For an example in Figure 7.3(a), the communication cost value of “BC” at 256KB L2 cache size per core is 6.30×10^6, and it increases to 7.15×10^6 when the L2 cache size is decreased to 128KB. However, if we use “STKDRAM+RSAPM+HNOC”, “STKDRAM”, “STKDRAM+RSAPM” and “STKDRAM+HNOC” cases then this increased communication cost value of “BC” (7.15×10^6 at 128KB L2 cache per core) reduced to 3.26×10^6, 5.77×10^6, 5.09×10^6 and 3.48×10^6 respectively.

Figure 7.3 shows that cases “STKDRAM+RSAPM+HNOC”, “STKDRAM”, “STKDRAM+RSAPM” and “STKDRAM+HNOC” reduces the communication cost by an average of 51.87%, 15.42%, 28.33%, and 48.97% respectively as compared to the communication cost value of base-case (“BC”). “STKDRAM+RSAPM+HNOC” case uses the benefits of hybrid interconnects as well as self-adaptive page mapping and shows the highest reduction by an average of 51.87% and maximum of 54.86%. (for ferret benchmark) from the communication cost value of “BC” for the 128KB size of the L2 cache per core.

Also, “STKDRAM+RSAPM+HNOC” shows an average of 36.45%, 23.54% and 2.9% more reduction in the value of “BC” (for the 128KB size of the L2 cache per core) as compared to the “STKDRAM”, “STKDRAM+RSAPM” and “STKDRAM+HNOC” respectively. Similarly, the case “STKDRAM+RSAPM+HNOC” takes the benefits of the hybrid interconnects along with the self-adaptive page mapping and reduces the increased value of the communication cost incurred due to the smaller size L2 cache per core.

Interestingly, for any particular L2 cache size (say 128KB or any other considered L2 cache size) and without reducing the L2 cache size per core, from Figure 7.3(a), 7.3(b), 7.3(c) and 7.3(d) we can see that “STKDRAM+RSAPM+HNOC” reduces the communication cost value of “BC” by an average of 50.69%. Whereas, “STKDRAM”,

TH-2297_136201001

"STKDRAM+RSAPM" and "STKDRAM+HNOC" reduces the communication cost value of "BC" by an average of 16.58%, 28.35% and 47.70% respectively.

Figure 7.4(a), 7.4(b), 7.4(c), and 7.4(d) shows that IPC value is very small for "BC". Also, if the L2 cache size per core is reduced from a larger value to a smaller, then the IPC value of the "BC" reduces to further smaller values. IPC values of the "STKDRAM", "STKDRAM+RSAPM", "STKDRAM+HNOC", and "STKDRAM+RSAPM+HNOC" cases also reduces as compared to their higher IPC values associated with the larger L2 cache size per core. Figure 7.4, shows that the IPC value of the "STKDRAM", "STKDRAM+RSAPM", "STKDRAM+HNOC", and "STKDRAM+RSAPM+HNOC" cases perform better as compared to the IPC value of the "BC" by an average of 145%, 160% 185.45%, and 187.27% respectively.

For an example in Figure 7.4(a), the IPC value of "BC", "STKDRAM", "STKDRAM+RSAPM", "STKDRAM+HNOC", and "STKDRAM+RSAPM+HNOC" at 128KB L2 cache size per core is 0.59, 1.45, 1.48, 1.62, and 1.63 respectively. Whereas, at 32 KB L2 cache size per core, the IPC value of "BC", "STKDRAM", "STKDRAM+RSAPM", "STKDRAM+HNOC", and "STKDRAM+RSAPM+HNOC" is 0.39, 1.18, 1.25, 1.42, and 1.44 respectively. Therefore, we can see that "STKDRAM+RSAPM+HNOC" at 32 KB L2 cache size per core performs almost similar to the "STKDRAM" case and better than "BC" case at the 128KB L2 cache size per core. So, for the fixed die size and to increase the core count, "STKDRAM+RSAPM+HNOC" improves the chip multiprocessor performance even after the reduction in the cache size per core.

Therefore, based on the result and design specifications, consider $config(a)$: associated with a 64-core CMP system having 1 GB 3D-stacked DRAM (overall 4GB), 4 optical interconnect based hybrid NOC, and 64KB L2 cache per core along with the page mapping technique, and $config(b)$: associated with a 64-core CMP system having overall 4GB off-chip DRAM, only 2D-mesh based NOC, and 128KB L2 cache per core, we can say that $config(a)$ performs better than $config(b)$.

7.6 Performance and Area Overheads

7.6.1 Performance Overheads

In almost all the contemporary CMPs, the standard size of the message (or packet) for network-on-chip (NOC) is 64 bytes; therefore, 64 packets are needed to transfer

TH-2297_136201001

Table 7.5: Example: Total number of L2 cache misses and page migrations.

Benchmarks	# L2 Misses	# Migrations	Migration %
barnes	1.03×10^6	3.70×10^4	3.59%
ferret	1.80×10^6	3.51×10^3	0.19%
fmm	1.22×10^6	8.65×10^3	0.70%
x264	6.87×10^5	2.78×10^3	0.40%

a 4KB page. Moreover, the maximum page migration cost associated with each page is 64 times the maximum distance between the source memory slice and destination memory slices. Insertion of high bandwidth, low latency optical interconnects between memory slices reduce the page migration time significantly. Also, there is no latency overhead as the page migration happens in parallel with the regular application execution.

Table 7.5 shows the percentage of the overall number of page migrations calculated with respect to the total number of L2 cache misses (for L2 cache size=128KB). It shows that migration due to run-time adaptive page mapping incurs on an average 1.22% overhead as compared to the total number of L2 cache misses. Moreover, for simplicity and due to the two order smaller number of page migrations as compared to the total L2 cache misses (as shown in Table 7.5), we have not considered the network contention due to the page migrations.

7.6.2 Area Overhead

For a 4GB (G_{size}= 4GB) DRAM memory with 4KB page size and having 4 (M= 4) DRAM slices, there are $\frac{4GB}{4KB}$ number of physical pages. Therefore, we require $\frac{4GB}{4KB} \times 2 \times 4$ number of 16 bit counters, $(M+1)$ comparisons, M additions, and M subtractions for all the DRAM memory pages. For 4 ($M = 4$) DRAM slices, the modified DC incurs a hardware overhead of 4 adders and comparators. Also, for migration list Mig, we need an overall of $\frac{4GB}{4KB}$ bit or 128KB of memory space. Similarly, location array $Loc[VP]$ needs an overall of $2\times\frac{4GB}{4KB}$ bit or 256KB of memory space (2-bit to represent four DRAM slices).

Therefore, to implement run-time page mapping, profiling counters along with the location array Loc and migration list Mig needs 16MB + 256KB + 128KB memory space associated with the overall 4GB of DRAM memory. Whereas, 4GB DRAM memory as cache, needs 256MB space to maintain the coherence directory for the 64-byte cache block size (26 bits for block identification ($\frac{4GB}{64B} = \frac{2^{32}}{2^6} = 2^{26}$ cache blocks),

TH-2297_136201001

2 bits to maintain coherence and 2 bits for DRAM slices per cache block) [30, 32, 36]. Therefore, the area overhead of the DRAM as a cache is more.

Also, for 8×8 CMP system having four ONIs, VIA table incurs 3-bits of memory to represent each intermediate nodes (as the number of ONIs are four and we also need to represent the -1 (NULL) value). Therefore, overall VIA table incurs an overhead of 64×(6 bit for destination+3 bit for first intermediate node+3 bit for second intermediate node) bits at each router.

7.7 Summary

In this work, we have analyzed the trade-off between the CMP performance (IPC and communication cost due to the memory page access) and cache size per core (L2 cache in our case). We found that for a fixed CMP die size, to increase the core count of the CMP reducing the cache size per core (L2 cache or LLC in our case) increases the on-chip communication cost and decreases the system IPC. However, the CMP performance degradation due to the smaller cache per core can be enhanced with the use of an efficient hybrid interconnection network, 3D-stacked DRAM memory, and an efficient application mapping. Also, the hybrid interconnection network and 3D-stacked DRAM memory fulfill the need of the CMP with the higher core count.

We found that cache size per core can be reduced up to a certain extent while using the efficient hybrid interconnection network, 3D-stacked DRAM memory, and a run-time self-adaptive application mapping. Also, for a given cache size per core (when not reducing the cache size), the use of the efficient hybrid interconnection network, 3D-stacked DRAM memory, and application mapping increases the system IPC and decreases the on-chip communication cost significantly.

In this thesis, we have assumed that queuing delay is having negligible effects on system performance as compared to the latency delay due to the interconnects (on-chip and off-chip). Moreover, the same is supported by the benchmark summary Table 7.5 for the used benchmarks, which shows a low injection rate (< 20). Therefore, we did not model the queuing delay at the junction points (of the electrical and optical interconnects). The used Sniper simulator considers (models) the effects of the "queuing delay" at the electrical 2D-mesh network. Our results presented in this thesis are produced using the simulator, which considers and models the effects of the queuing delay for the electrical 2D-mesh network. Also, as the research related to

TH-2297_136201001

the self-adaptive run-time page mapping considering the 3D-stacked memory-based CMPs with/without hybrid interconnect is in a nascent stage. Therefore, as a foundational and preliminary work, we have modeled the access latencies without considering the queuing delays.

Further, considering the "queuing delay" at the time of latency modeling (in Section 7.2) at the junction points of the electrical and optical interconnects is an essential future work of this Chapter for the high injection rate (MPKI) applications. As application having a high injection rate (MPKI) may create queuing delays and may not be ignored while modeling the latency equations. Also, analyzing the effect of different core size on to the CMP system performance is an important future work that needs to be explored, as the higher number of cores may decrease the T_{cal} value for the highly parallel applications that in turn may reduce the system IPC.

TH-2297_136201001

8

Conclusions and Future Perspectives

In the multi-core era, technology advancement has led the developments of the chips that can have up to hundreds of cores on a single chip [12, 107]. The number of cores on a single chip is also expected to go beyond hundred to satisfy the growing need of the current as well as future applications [17, 7]. However, there are several architectural constraints and resource management techniques that limit the benefits of the increasing core count. Researchers have found that bandwidth limitations of the off-chip memory and on-chip interconnects are the primary constraints that severely restricts the CMP core count and thereby performance benefits [100, 50, 7].

Researchers have proposed many high bandwidth 3D-stacked memories such as DRAM memory and non-volatile memories (NVMs) (including phase-change memory (PCM), magnetic random access memories (MRAM), etc.), to address the memory bandwidth and performance demands of the current as well as future CMP systems [69, 55, 30, 126]. Moreover, NOC has emerged as a viable alternative to fulfill the present and future demand of the modular and scalable interconnection network [13]. Recently, researchers have started exploring many efficient on-chip interconnection networks, including optical and wireless NOC, to satisfy the growing need of the CMP systems [35, 123, 61].

Assimilating the past and the needs of the present as well as future CMP systems, the contributions made in this thesis addresses the importance and challenges related to the 3D-stacked memory and network interconnect architectures that have been

TH-2297_136201001

proposed for the current as well as future CMP systems. This chapter presents a summary of the contributions made in this thesis. This chapter concludes with the avenues for future research and directions of the extension.

8.1 Summary of Thesis

This thesis aims to introduce the methods of application mapping considering the current as well as future generation CMP system architectural variations. It is envisaged that the contributions made in this thesis can be appropriately applied to the future CMP systems where state-of-art methods may seize to apply.

With a bottom-up approach, the first contribution (Chapter 4) laid a foundation of the application mapping, and the conclusions arrived from this are used in the next contributions of this thesis. In the first contribution, a static profile based multi-threaded application mapping (using different types of thread to the core and virtual page to DRAM slice mapping) has been performed for the 3D-stacked DRAM memory based target CMP. Experiments show that the overall on-chip communication cost reduction due to the page mapping is significantly higher as compared to the reduction due to the thread mapping. Moreover, virtual page to DRAM slice mapping and thread to core mapping reduces overall on-chip communication cost up to 86% (average 56%) and 26% (average 12%) respectively.

The conclusion of the first contribution (virtual page mapping is more effective as compared to the thread mapping) along with the facts (a) thread migration is a costlier operation and (b) most of the application shows phase-wise behavior at the run-time, prompted us to propose a self-adaptive run-time page mapping technique in the second contribution (Chapter 5) of this thesis. Further, in the second contribution (Chapter 5), we have performed the comparison between proposed method along with an improvement (addition of SRAM mapping buffer) and a recent state of work (as proposed in [30]). Our experimental result shows that the proposed method can be *an alternative way to use the 3D-stacked DRAM memory for current as well as future CMP systems.* The proposed self-adaptive run-time page mapping alone shows the communication cost reduction up to a maximum of 80% and an average of about 40% as compared to the base case method. Further, our self-adaptive run-time page mapping together with the SRAM mapping buffer outperforms the base-case by an average of 48% in terms of overall execution time. Also, most importantly,

TH-2297_136201001

the adaptive run-time mapping with the SRAM mapping buffer shows a performance improvement by an average of 40% (in terms of overall execution time) when compared to 3D-stacked DRAM used as a coherent cache with temporal SRAM buffer, a state-of-art work proposed by [30].

As the future, CMPs (where the number of cores is expected to increase) needs much larger memory bandwidth for the better performance, and research has shown that increasing the DRAM memory size incurs much DRAM refresh related power consumption [79]. Therefore, for the future CMPs, 3D-stacked DRAM-PCM hybrid memory is proposed as a viable alternative of the 3D-stacked DRAM memory and under exploration. Consequently, in third contribution (Chapter 6) of this thesis, we have considered a 3D-stacked hybrid DRAM-PCM memory based target CMP system and to take advantage of the large capacity 3D-stacked hybrid memory as well as to minimize the DRAM refresh operations (its associated power consumption overhead) and remote memory accesses overheads, we performed a simple DRAM access-aware page placement technique between DRAM and PCM of the hybrid memory slice. Further, our method uses the DRAM row access information and performs an access-aware self-adaptive page mapping for the optimized page placement between the different hybrid memory modules of the 3D-stacked hybrid memory. In this way, our method provides an alternative and efficient way to use 3D-stacked hybrid DRAM-PCM memory for the future generation CMP system. Our proposed approach performs similar or better in terms of the execution time as compared to the base case. However, the proposed technique (in Chapter 6) reduces the energy consumption due to the DRAM refresh by an average of 51% as compared to the base case.

The fourth and final contribution of this thesis (Chapter 7) performs the trade-off analysis between the performance and cache size of the CMP system while utilizing the benefits of the high-end optical interconnects, 3D-stacked DRAM memory and a self-adaptive run-time page mapping. The study has shown that for a fixed-size die, increasing the number of cores on a chip reduces the on-chip caches per core [7]. However, on-chip caches are one of the vital parameters to get better performance, and a shortage of caches may degrade the system performance. In this final contribution (Chapter 7), we found that for a fixed CMP die size, reducing the cache size per core (L2 cache or LLC in our case) increases the on-chip communication cost and decreases the system IPC. However, the CMP performance degradation due to the smaller cache

TH-2297_136201001

per core can be enhanced with the use of an efficient hybrid interconnection network (a combination of electrical and optical interconnects), 3D-stacked DRAM memory, and our self-adaptive run-time page mapping. Also, the hybrid interconnection network and 3D-stacked DRAM memory fulfill the need of the future CMP systems.

Our result analysis in fourth contribution (Chapter 7) has shown that cache size per core can be reduced up to a certain extent while using the efficient hybrid interconnection network, 3D-stacked DRAM memory, and a run-time self-adaptive application mapping. Also, for a given cache size per core (when not reducing the cache size), the use of the efficient hybrid interconnection network, 3D-stacked DRAM memory, and application mapping increases the system IPC and decreases the on-chip communication cost significantly.

8.2 Future Research Avenues

The contributions made in the chapters of this thesis provide ample scope and several clear directions for future research. Though the *application mapping (mainly due to the page mapping)*, the second contribution of this thesis is still in its nascent form for the modern CMP systems having larger core count, it has a strong potential to use the modern architectural designs (such as 3D-stacked memories and high-end NOCs) in an alternative and efficient way. One immediate upgrade to the *self-adaptive run-time page mapping* could be the use of any machine learning approach to decide the Cor_{Thr} value (which is the deciding parameter for the number of page migration) at the run-time, instead of a fixed Cor_{Thr} value.

To perform the self-adaptive run-time page mapping, we need overall of 16MB memory space for the profiling counters, which is very less if we compare with the storage overhead associated with the 256MB coherence directory for the 4GB 3D-stacked DRAM as a cache with 64-byte cache block size (as explained in [32, 36]). However, this 16MB space associated with the profiling counters is still huge in itself. Therefore, another dimension of future work can be to reduce the considerable amount of the counter space by adopting further efficient counter design such as given by Zhao *et al.* in [128]. Moreover, as a new dimension, we can further reduce the amount of counter space by adopting a new mapping technique that selects only highly active pages (applying a filter on the active pages).

In this thesis, we have used page access behavior of the threads to perform the

TH-2297_136201001

efficient phase-wise mapping. However, we may use many other profile information to perform the thread/page mapping, which may lead to other future dimensions of this thesis.

Also, for the dynamic run-time mapping, we may use the combination of thread and page mapping techniques, where any recent less overhead based thread to core mapping technique can be used [120].

TH-2297_136201001

TH-2297_136201001

Publications

- **Rakesh Pandey** and Aryabartta Sahu, "Run-time adaptive data page mapping: A Comparison with 3D-stacked DRAM cache", in *Elsevier Journal of Systems Architecture (JSA), 2020.*

- **Rakesh Pandey** and Aryabartta Sahu, "Performance and Area Trade-off of 3D-stacked DRAM Based Chip Multiprocessor with Hybrid Interconnect," in *IEEE Transactions on Emerging Topics in Computing (TETC), 2019* (Early Access, DOI: 10.1109/TETC.2019.2946887).

- **Rakesh Pandey** and Aryabartta Sahu, "Access-Aware Self-adaptive Data Mapping onto 3D-Stacked Hybrid DRAM-PCM based Chip-Multiprocessor", *The 21st IEEE International Conference on High Performance Computing and Communications (HPCC), 2019.*

- **Rakesh Pandey** and Aryabartta Sahu, "Efficient Mapping of Multi-threaded Applications onto 3D Stacked Chip-Multiprocessor", *The 19th IEEE International Conference on High Performance Computing and Communications (HPCC), 2017.*

- **Rakesh Pandey** and Aryabartta Sahu, "Adaptive Multi-workload Mapping onto 3D Stacked Chip-Multiprocessor having Multiple Memory Controllers and Channels", (**Manuscript in Preparation**).

- **Rakesh Pandey** and Aryabartta Sahu, "Performance Analysis considering Multi-workload Mapping onto 3D Stacked Chip-Multiprocessor having Multiple optical links and Memory Controllers", (**Manuscript in Preparation**).

TH-2297_136201001

TH-2297_136201001

Vitae

Rakesh Pandey joined the Dual (M.Tech + Ph.D) Degree programme at the Department of Computer Science and Engineering (CSE) of Indian Institute of Technology (IIT) Guwahati, India in July 2013. Prior to joining Dual Degree, he did his Bachelors of Technology (B.Tech) degree in Electronics and Communication Engineering (ECE) from Uttar Pradesh Technical University, Lucknow, India.

He has keen interests in pursuing the field of computer architecture. His current research interests include DRAM memory controller design for the adaptive application mapping, and consideration of the high end optical interconnects in the NOC design for the 3D-stacked chip multiprocessor system. He enjoys playing cricket, watching scientific movies and traveling to the places where nature has bestowed its bounty.

Contact Information

Email : rakesh.pandey@iitg.ac.in,
rakeshpandey377@gmail.com

Address : Village- Kataya, P.O.- Udairajgang
Distt.-Siddharthnagar, Uttar Pradesh- 272204,
INDIA

TH-2297_136201001

Bibliography

[1] AMD Ryzen™ Threadripper™ Processors. https://www.amd.com/en/products/ryzen-threadripper. Accessed: 2019-09-13.

[2] An Intro to MCDRAM (High Bandwidth Memory) on Knights Landing. https://software.intel.com/en-us/blogs/2016/01/20/an-intro-to-mcdram-high-bandwidth-memory-on-knights-landing. Accessed: 2019-08-15.

[3] Intel's First Microprocessor. https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html. Accessed: 2019-08-15.

[4] Snapdragon 8 Series Mobile Platforms. https://www.qualcomm.com/products/snapdragon-8-series-mobile-platforms. Accessed: 2019-09-13.

[5] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti. Achieving Predictable Performance Through Better Memory Controller Placement in Many-core CMPs. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 451–461, 2009.

[6] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas. Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 400–411, Feb 2013.

[7] N. B. Anant Agarwal, Jason Miller and D. Wentzlaff. Core Count vs Cache Size for Manycore Architectures in the Cloud. In *CSAIL Technical Reports*, volume MIT-CSAIL-TR-2010-008, pages 39–50, Feb 2010.

[8] S. Bahirat and S. Pasricha. Exploring Hybrid Photonic Networks-on-chip Foremerging Chip Multiprocessors. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '09, pages 129–136, 2009.

[9] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA '05, pages 506–517, 2005.

TH-2297_136201001

[10] J. Balfour and W. J. Dally. Design Tradeoffs for Tiled CMP On-chip Networks. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS '06, pages 187–198, 2006.

[11] S. Banerjee, G. Surendra, and S. K. Nandy. On the Effectiveness of Phase Based Regression Models to Trade Power and Performance Using Dynamic Processor Adaptation. *Journal of Systems Architecture*, 54(8):797–815, Aug. 2008.

[12] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, and *et al.* TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 88–598, Feb 2008.

[13] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[14] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, 2008.

[15] T. Bjerregaard and S. Mahadevan. A Survey of Research and Practices of Network-on-chip. *ACM Comput. Surv.*, 38(1), June 2006.

[16] L. Bononi and N. Concer. Simulation and analysis of network on chip architectures: Ring, spidergon and 2d mesh. In *Proceedings of the Conference on Design, Automation and Test in Europe: Designers' Forum*, DATE '06, pages 154–159, 2006.

[17] S. Borkar. Thousand Core Chips: A Technology Perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749, 2007.

[18] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao. Emerging NVM: A Survey on Architectural Integration and Research Challenges. *ACM Trans. Des. Autom. Electron. Syst.*, 23(2):14:1–14:32, Nov. 2017.

[19] P. Brucker. *Scheduling Algorithms*. Springer, 5th edition, 2010.

[20] B. Bryan, A. Murali, B. Ned, D. John, Jiang, and *et al.* Die Stacking (3D) Microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 469–479, 2006.

[21] X. Bu, J. Rao, and C.-z. Xu. Interference and Locality-aware Task Scheduling for MapReduce Applications in Virtual Clusters. In *Proceedings of the 22Nd*

TH-2297_136201001

International Symposium on High-performance Parallel and Distributed Computing, HPDC '13, pages 227–238, 2013.

[22] D. R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[23] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout. An Evaluation of High-Level Mechanistic Core Models. *ACM Transactions on Architecture and Code Optimization*, 11(3):28:1–28:25, Aug. 2014.

[24] S. Chakraborty and H. K. Kapoor. Analysing the Role of Last Level Caches in Controlling Chip Temperature. *IEEE Transactions on Sustainable Computing*, 3(4):289–305, Oct 2018.

[25] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens. System and circuit level power modeling of energy-efficient 3D-stacked wide I/O DRAMs. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 236–241, March 2013.

[26] C. Chang and Y. Hsu. Object oriented network-on-chip modeling. In *2010 International Computer Symposium (ICS2010)*, pages 457–466, Dec 2010.

[27] G. Chen, F. Li, S. W. Son, and M. Kandemir. Application Mapping for Chip Multiprocessors. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 620–625, 2008.

[28] Y.-J. Chen, C.-L. Yang, and J.-J. Chen. Distributed Memory Interface Synthesis for Network-on-chips with 3D-stacked DRAMs. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '12, pages 458–465, 2012.

[29] C. B. Cho and T. Li. Complexity-based Program Phase Analysis and Classification. In *2006 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 105–113, Sept 2006.

[30] C. Chou, A. Jaleel, and M. K. Qureshi. CANDY: Enabling coherent DRAM caches for multi-node systems. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13, Oct 2016.

[31] C. Chou and R. Marculescu. Run-Time Task Allocation Considering User Behavior in Embedded Multiprocessor Networks-on-Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(1):78–91, Jan 2010.

TH-2297_136201001

[32] B. A. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. F. Duato. Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks. In *Proc. of the ISCA*, pages 93–104, 2011.

[33] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, June 2001.

[34] B. K. Daya, C. O. Chen, S. Subramanian, W. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L. Peh. SCORPIO: A 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC with in-network ordering. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 25–36, 2014.

[35] S. Deb, K. Chang, X. Yu, S. P. Sah, M. Cosic, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo. Design of an Energy-Efficient CMOS-Compatible NoC Architecture with Millimeter-Wave Wireless Interconnects. *IEEE Transactions on Computers*, 62(12):2382–2396, Dec 2013.

[36] S. Demetriades and S. Cho. Stash directory: A scalable directory for many-core coherence. In *2014 IEEE 20th International Symposium on HPCA*, pages 177–188, Feb 2014.

[37] W. Ding, Y. Zhang, M. Kandemir, J. Srinivas, and P. Yedlapalli. Locality-aware mapping and scheduling for multicores. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–12, Feb 2013.

[38] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu. MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 1–10, May 2012.

[39] A. Fawibe, J. Sherman, K. Kavi, M. Ignatowski, and D. Mayhew. New Memory Organizations for 3D DRAM and PCMs. In *Architecture of Computing Systems*, pages 200–211. Springer Berlin Heidelberg, 2012.

[40] Feihui Li, C. Nicopoulos, T. Richardson, Yuan Xie, V. Narayanan, and M. Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 130–141, June 2006.

[41] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Hetero-

TH-2297_136201001

geneous Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(6):911–924, June 2010.

[42] M. Frigo, C. E. Leiserson, and K. H. Randall. The Implementation of the Cilk-5 Multithreaded Language. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI '98, pages 212–223, 1998.

[43] E. Fusella and A. Cilardo. H2ONoC: A Hybrid Optical–Electronic NoC Based on Hybrid Topology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1):330–343, Jan 2017.

[44] M. Ghosh and H. S. Lee. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 134–145, Dec 2007.

[45] J. Goodacre and A. N. Sloss. Parallelism and the ARM instruction set architecture. *Computer*, 38(7):42–50, July 2005.

[46] M. Guan and L. Wang. Improving DRAM Performance in 3-D ICs via Temperature Aware Refresh. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(3):833–843, March 2017.

[47] Hai Jiang and V. Chaudhary. Compile/run-time support for thread migration. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 9 pp–, April 2002.

[48] S. M. Hassan, S. Yalamanchili, and S. Mukhopadhyay. Near Data Processing: Impact and Optimization of 3D Memory System Architecture on the Uncore. In *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, pages 11–21, 2015.

[49] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *Computer*, 41(7):33–38, July 2008.

[50] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, Apr 2001.

[51] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27(5):51–61, Sept. 2007.

[52] Jaehyuk Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. In *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, Sep. 2001.

TH-2297_136201001

[53] A. Jantsch and H. Tenhunen, editors. *Networks on Chip*. Kluwer Academic Publishers, 2003.

[54] J. Jeffers and J. Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc., 1st edition, 2013.

[55] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 25–37, Dec 2014.

[56] Jingcao Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, April 2005.

[57] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, and H. L. *et al.* 8Gb 3D DDR3 DRAM using through-silicon-via technology. In *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 130–131,131a, Feb 2009.

[58] J. Kim, J. Balfour, and W. Dally. Flattened Butterfly Topology for On-Chip Networks. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 172–182, 2007.

[59] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, Dec. 1999.

[60] P. S. Laursen. Simulated Annealing for the QAP- Optimal Tradeoff between Simulation Time and Solution Quality. *European Journal of Operational Research*, 69(2):238 – 243, 1993.

[61] S. Le Beux, H. Li, I. O'Connor, K. Cheshmi, X. Liu, J. Trajkovic, and G. Nicolescu. Chameleon: Channel efficient Optical Network-on-Chip. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[62] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4):63:1–63:29, Jan. 2016.

[63] E. Lee, J. E. Jang, T. Kim, and H. Bahn. On-demand snapshot: An efficient versioning file system for phase-change memory. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2841–2853, Dec 2013.

TH-2297_136201001

[64] H. G. Lee, S. Baek, C. Nicopoulos, and J. Kim. An Energy- and Performance-aware DRAM Cache Architecture for Hybrid DRAM/PCM Main Memory Systems. In *Proceedings of the 2011 IEEE 29th International Conference on Computer Design*, ICCD '11, pages 381–387, 2011.

[65] M. Li, Q.-A. Zeng, and W.-B. Jone. DyXY - a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 849–852, July 2006.

[66] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 315–326, 2008.

[67] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari. Bridging the Processor-Memory Performance Gapwith 3D IC Technology. *IEEE Design and Test of Computers*, 22(6):556–564, Nov. 2005.

[68] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 1–12, 2012.

[69] G. H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 453–464. IEEE Computer Society, June 2008.

[70] G. L. Loi, B. Agrawal, N. Srivastava, Sheng-Chih Lin, T. Sherwood, and K. Banerjee. A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 991–996, July 2006.

[71] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi. Scale-out Processors. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 500–511, 2012.

[72] D. Lustig, A. Bhattacharjee, and M. Martonosi. TLB Improvements for Chip Multiprocessors: Inter-Core Cooperative Prefetchers and Shared Last-Level TLBs. *ACM Transactions on Architecture and Code Optimization*, 10(1):2:1–2:38, Apr. 2013.

[73] R. Marculescu, U. Y. Ogras, L. Peh, N. E. Jerger, and Y. Hoskote. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1):3–21, Jan 2009.

TH-2297_136201001

[74] A. K. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. R. Das. Architecting On-chip Interconnects for Stacked 3D STT-RAM Caches in CMPs. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 69–80, 2011.

[75] S. Mittal and J. S. Vetter. A Survey Of Techniques for Architecting DRAM Caches. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1852–1863, June 2016.

[76] M. E. Mortenson. *Geometric Modeling*. John Wiley & Sons, Inc., 1985.

[77] A. Muddukrishna, P. A. Jonsson, and M. Brorsson. Locality-aware Task Scheduling and Data Distribution for OpenMP Programs on NUMA Systems and Manycore Processors. *Sci. Program.*, 2015:5:5–5:5, Jan. 2016.

[78] S. Murali, S. Murali, G. De Micheli, G. De Micheli, and G. De Micheli. Bandwidth-Constrained Mapping of Cores Onto NoC Architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, DATE '04, pages 20896–, 2004.

[79] O. Mutlu and L. Subramanian. Research Problems and Opportunities in Memory Systems. *Supercomput. Front. Innov.: Int. J.*, 1(3):19–55, Oct. 2014.

[80] S. Nafiul, B. Hameed, and C. Jeanine. LMStr: An On-Chip Shared Hardware Controlled Scratchpad Memory for Multicore Processors. *MemSys*, 7 2017.

[81] P. Nair, C. Chou, and M. K. Qureshi. A case for Refresh Pausing in DRAM memory systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 627–638, Feb 2013.

[82] T. Oh, H. Lee, K. Lee, and S. Cho. An Analytical Model to Study Optimal Area Breakdown between Cores and Caches in a Chip Multiprocessor. In *2009 IEEE Computer Society Annual Symposium on VLSI*, pages 181–186, May 2009.

[83] K. Olukotun, L. Hammond, and J. Laudon. *Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency*. Morgan and Claypool, 2007.

[84] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The Case for a Single-chip Multiprocessor. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VII, pages 2–11, 1996.

[85] S. Onsori, A. Asad, K. Raahemifar, and M. Fathy. High performance 3D CMP design with stacked hybrid memory architecture in the dark silicon era using a convex optimization model. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2607–2610, May 2016.

TH-2297_136201001

[86] I. O'Connor and G. Nicolescu. *Integrated Optical Interconnect Architectures for Embedded Systems.* Springer-Verlag New York, 01 2013.

[87] P. S. Pacheco, editor. Morgan Kaufmann, 2011.

[88] R. Pandey and A. Sahu. Efficient Mapping of Multi-threaded Applications onto 3D Stacked Chip-Multiprocessor. In *2017 IEEE 19th International Conference on High Performance Computing and Communications (HPCC)*, pages 324–331, Dec 2017.

[89] R. Pandey and A. Sahu. Access-Aware Self-adaptive Data Mapping onto 3D-stacked Hybrid DRAM-PCM based Chip-Multiprocessor. In *The 21st IEEE International Conference on High Performance Computing and Communications (HPCC-2019)*, Aug. 2019.

[90] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., 1982.

[91] B. Pham, A. Bhattacharjee, Y. Eckert, and G. H. Loh. Increasing TLB reach by exploiting clustering in page translations. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 558–567, Feb 2014.

[92] T. Q. Pham and P. K. Garg. *Multithreaded Programming with Win32.* Prentice Hall, 1st edition, 1998.

[93] B. Pourshirazi and Z. Zhu. Refree: A Refresh-Free Hybrid DRAM/PCM Main Memory System. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 566–575, May 2016.

[94] Qiaosha Zou, M. Poremba, Rui He, Wei Yang, J. Zhao, and Y. Xie. Heterogeneous architecture design with emerging 3D and non-volatile memory technologies. In *The 20th Asia and South Pacific Design Automation Conference*, pages 785–790, Jan 2015.

[95] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 24–33. ACM, 2009.

[96] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page Placement in Hybrid Memory Systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 85–95, 2011.

TH-2297_136201001

[97] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. . Chen, R. M. Shelby, M. Salinga, D. Krebs, S. . Chen, H. . Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465–479, July 2008.

[98] M. Reshadi, A. Khademzadeh, A. Reza, and M. Bahmani. A novel mesh architecture for on-chip networks. *D & R Industry Articles, http://www. design-reuse. com/articles/23347/on-chipnetwork. html*, 2013.

[99] R. Rodríguez-Rodríguez, F. Castro, D. Chaver, L. Pinuel, and F. Tirado. Reducing Writes in Phase-change Memory Environments by Using Efficient Cache Replacement Policies. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 93–96, 2013.

[100] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 371–382, 2009.

[101] E. Salminen, A. Kulmala, and T. D. Hamalainen. On network-on-chip comparison. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, pages 503–510, Aug 2007.

[102] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis. An Analysis of On-chip Interconnection Networks for Large-scale Chip Multiprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(1):4:1–4:28, May 2010.

[103] A. Shacham, K. Bergman, and L. P. Carloni. On the Design of a Photonic Network-on-Chip. In *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, pages 53–64, 2007.

[104] X. Shen, Y. Zhong, and C. Ding. Locality Phase Prediction. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, pages 165–176, 2004.

[105] T. Sherwood, S. Sair, and B. Calder. Phase Tracking and Prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 336–349, 2003.

[106] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak. A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems. *ACM Comput. Surv.*, 50(2):24:1–24:40, Apr. 2017.

TH-2297_136201001

[107] A. Sodani, R. Gramunt, J. Corbal, H. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. Liu. Knights landing: Second-generation intel xeon phi product. *IEEE Micro*, 36(2):34–46, Mar 2016.

[108] S. Sreepathi, E. D'Azevedo, B. Philip, and P. Worley. Communication Characterization and Optimization of Applications Using Topology-Aware Task Mapping on Large Supercomputers. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 225–236, 2016.

[109] V. Suhendra, C. Raghavan, and T. Mitra. Integrated Scratchpad Memory Optimization and Task Scheduling for MPSoC Architectures. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 401–410, 2006.

[110] Tao Zhang, Kui Wang, Yi Feng, Xiaodi Song, Lian Duan, Y. Xie, Xu Cheng, and Youn-Long Lin. A customized design of DRAM controller for on-chip 3D DRAM stacking. In *IEEE Custom Integrated Circuits Conference 2010*, pages 1–4, Sep. 2010.

[111] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM Design and Organization for Energy-constrained Multi-cores. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 175–186, 2010.

[112] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn. Corona: System Implications of Emerging Nanophotonic Technology. In *2008 International Symposium on Computer Architecture*, pages 153–164, June 2008.

[113] S. Werner, J. Navaridas, and M. Luján. A Survey on Optical Network-on-Chip Architectures. *ACM Comput. Surv.*, 50(6):89:1–89:37, Dec. 2017.

[114] S. Werner, J. Navaridas, and M. Luján. Designing Low-Power, Low-Latency Networks-on-Chip by Optimally Combining Electrical and Optical Links. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 265–276, Feb 2017.

[115] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor System-on-Chip (MPSoC) Technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, Oct 2008.

[116] H. . P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase Change Memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec 2010.

TH-2297_136201001

[117] D. H. Woo, N. H. Seong, D. L. Lewis, and H. S. Lee. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–12, Jan 2010.

[118] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22Nd Annual International Symposium on Computer Architecture*, ISCA '95, pages 24–36, 1995.

[119] L. Wu and W. Zhang. Cache-aware SPM allocation algorithms for hybrid SPM-cache architectures. *Sixteenth International Symposium on Quality Electronic Design*, pages 123–129, 2015.

[120] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 595–610, USA, 2018. USENIX Association.

[121] L. Yavits, A. Morad, and R. Ginosar. 3D cache hierarchy optimization. In *2013 IEEE International 3D Systems Integration Conference (3DIC)*, pages 1–5, Oct 2013.

[122] L. Yavits, A. Morad, and R. Ginosar. Cache Hierarchy Optimization. *IEEE Computer Architecture Letters*, 13(2):69–72, July 2014.

[123] Y. Ye, J. Xu, B. Huang, X. Wu, W. Zhang, X. Wang, M. Nikdast, Z. Wang, W. Liu, and Z. Wang. 3-D Mesh-Based Optical Network-on-Chip for Multiprocessor System-on-Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(4):584–596, April 2013.

[124] H. Yoon. Row Buffer Locality Aware Caching Policies for Hybrid Memories. In *Proceedings of the 2012 IEEE 30th International Conference on Computer Design (ICCD 2012)*, ICCD '12, pages 337–344, 2012.

[125] S. Yu and P. Chen. Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-State Circuits Magazine*, 8(2):43–56, Spring 2016.

[126] W. Zhang and T. Li. Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, PACT '09, pages 101–112, 2009.

TH-2297_136201001

[127] Y. Zhang, L. Li, Z. Lu, A. Jantsch, M. Gao, H. Pan, and F. Han. A Survey of Memory Architecture for 3D Chip Multi-processors. *Microprocess. Microsyst.*, 38(5):415–430, July 2014.

[128] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. *SIGMETRICS Perform. Eval. Rev.*, 34(1):323–334, June 2006.

[129] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 14–23, 2009.

[130] D. Zhu, Y. Li, and L. Chen. On Trade-off Between Static and Dynamic Power Consumption in NoC Power Gating. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[131] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek. Intel® QuickPath Interconnect Architectural Features Supporting Scalable System Architectures. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 1–6, Aug 2010.

TH-2297_136201001