Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo XXVIII

Settore Concorsuale di afferenza: 09/E3 Elettronica

Settore Scientifico disciplinare: ING-INF/01

Memory Hierarchy Design for Next Generation Scalable Many-core Platforms

Presentata da: **ERFAN AZARKHISH**

Coordinatore Dottorato	**Relatore**
Prof. Alessandro Vanelli Coralli	**Prof. Luca Benini**

Esame finale anno 2016

Memory Hierarchy Design for Next Generation Scalable Many-core Platforms

Erfan Azarkhish

a thesis submitted for the degree of

Doctor of Philosophy

at the University of Bologna, Bologna,

Italy.

February 23, 2016

I would like to dedicate this thesis to my loving wife.

Abstract

Performance and energy consumption in modern computing platforms is largely dominated by the memory hierarchy. The increasing computational power in the multiprocessors and accelerators, and the emergence of the data-intensive workloads (e.g. large-scale graph traversal and scientific algorithms) requiring fast transfer of large volumes of data, are two main trends which intensify this problem by putting even higher pressure on the memory hierarchy. This increasing gap between computation speed and data transfer speed is commonly referred as the "memory wall" problem.

With the emergence of heterogeneous Three Dimensional (3D) Integration based on through-silicon-vias (TSV), this situation has started to recover in the past years. On one hand, it is now possible to improve memory access bandwidth and/or latency by either stacking memories directly on top of processors or through abstracted memory interfaces such as Micron's Hybrid Memory Cube (HMC). On the other hand, near memory computation has become worthy of revisiting due to the cost-effective integration of logic and memory in 3D stacks. These two directions bring about several interesting opportunities including performance improvement, energy and cost reduction, product miniaturization, and modular design for improved time to market.

In this research, we study the effectiveness of the 3D integration technology and the optimization opportunities which it can provide in the different layers of the memory hierarchy in cluster-based many-core platforms ranging from intra-cluster L1 to inter-cluster L2 scratchpad memories (SPMs), as well as the main memory. We will demonstrate that with the current TSV technologies, moving towards the third dimension inside the processing

clusters can only be beneficial in terms of modularity, flexibility, and manufacturing cost. While, to achieve significant performance improvements, lower levels of the memory hierarchy should be explored.

In addition, by moving a part of the computation to where data resides, in the 3D-stacked memory context, we demonstrate further energy and performance improvement opportunities. Our obtained results are backed up by the physical implementation of cycle-accurate models (down to post place-and-route layouts) using industrial technology libraries, as well as, calibrated full-system simulation environments. We have used different industrial and academic platforms in synergy with each other to achieve accurate and realistic conclusions.

Acknowledgements

I would like to thank my supervisor Professor Luca Benini for his support and guidance over these years. He has provided exciting research opportunities for me, allowed me to meet brilliant people, and work in world-class institutions and research environments. I would like to thank my co-advisors Dr. Igor Loi and Dr. Davide Rossi for helping me during this research and for their valuable and insightful comments.

My gratitude also goes to everyone who has made this thesis possible: EMS Research Group at the University of Kaiserslautern for their valuable DRAM IP core, and their contribution to the developed DRAM controller; STMicroelectronics for providing access to one of their technology libraries; Samsung Electronics for their support and funding; Integrated Systems Laboratory at the ETH Zurich University for hosting me as guest researcher; and finally Christoph Pfister for his excellent work on address scrambling.

This thesis has been funded by European projects (VIRTICAL, Phidias, YINS, and Multitherman), and Samsung Electronics.

Contents

1 Introduction **1**
1.1 Organization of Dissertation 10

2 3D Stacking of L1 Scratchpad Memories **14**
2.1 Motivations and Challenges 14
2.2 Related Works 16
2.3 2D Logarithmic Interconnect 17
2.3.1 Network Protocol 18
2.3.2 Request Block 19
2.3.3 Response Block 20
2.4 3D Logarithmic Interconnect 20
2.4.1 Centralized 3D Logarithmic Interconnect 21
2.4.2 Distributed 3D Logarithmic Interconnect 22
2.5 Dealing with 3D Integration Issues 22
2.5.1 ESD Protection 23
2.5.2 Boot-time Configuration 23
2.5.3 Process/Voltage/Temperature Variations 24
2.6 Experimental Results 25
2.6.1 Comparison with Other Topologies 27
2.6.2 Design Alternatives 28
2.6.3 Discussion 32
2.7 Summary 32

3 3D Stacking of L2 Scratchpad Memories **34**
3.1 Motivations and Challenges 34
3.2 Related Works 36
3.3 3D-NUMA Memory IP 39
3.4 Network Operation 42
3.4.1 Role of the Read Buffer 43
3.4.2 Flow Control Components 43
3.5 Design Implementation 45
3.6 Performance Evaluation 48
3.6.1 Network Parameters of 3D-NUMA 49
3.6.2 3D-NUMA vs Memory Banks Attached to NoC 51
3.6.3 Effect of Memory Interleaving 52
3.6.4 Effect of Maximum Outstanding Transactions 54

3.6.5 Different Configurations with Equal Memory Size 55
3.7 Power and Temperature Analysis 56
3.7.1 Power Analysis 56
3.7.2 Thermal Analysis 59
3.8 Packaging and Power Delivery 64
3.9 Manufacturing Yield and Cost 65
3.10 Summary 68

4 Near Memory Computation in the L3 Memory Context 69
4.1 Motivations and Challenges 70
4.2 Related Works 71
4.3 The Smart Memory Cube (SMC) 74
4.3.1 The Main Interconnect on LoB 74
4.3.2 Address Remapping and Scrambling 76
4.3.3 Vault Controllers 81
4.4 Calibrating The CA Model 83
4.5 Experimental Results 85
4.5.1 HMC Exploration 86
4.5.2 Address Remapping 91
4.5.3 Handling PIM Traffic 96
4.6 Summary 100

5 Processor-in-Memory Design for the Smart Memory Cube 101
5.1 Motivations and Challenges 101
5.2 Related Works 103
5.3 The SMC Simulation Environment 104
5.4 Design of the Processor-in-Memory 106
5.4.1 PIM's Memory Model 106
5.4.2 Enhancing PIM's Functionality 108
5.5 Design of the Software Stack 109
5.5.1 Offloading Mechanisms 110
5.6 Experimental Results 111
5.6.1 Accuracy Verification and Calibration 111
5.6.2 Benchmarking Results 112
5.7 Summary 120

6 Conclusions 121

References 124

A Accuracy verification of the high-level SMC model 140

B Source codes of computation kernels 142

List of Tables

2.1 Comparison of different arbitration methods (p=16) 19
2.2 Performance comparison between LIN, NoC, and Bus executing different benchmarks. 28
2.3 Comparison of post-layout area between LIN and NoCs. 29

3.1 Number of power TSVs required for each die, and the total count for manufacturing one instance of each 65

5.1 Zero-load latency of memory accesses 112

List of Figures

1.1 Steady growth in the memory density over the years (Source:[8]) 2
1.2 Off-chip Bandwidth per Million Transistors at Peak Theoretical Clock Rate (Source:[8]) 2
1.3 Cross-section of the WIOMING Wide-IO DRAM (Source: [9]) (a) and the HMC's die photograph (Source: [10]) (b). 3
1.4 An overview of a typical memory hierarchy in the cluster-based many-core platforms. 4
1.5 A comparison of the computing efficiency among the biological brains and different computation technologies (Source:[11]). 6
1.6 Historical Clock-rates (Source:[8]). 6
1.7 Die photograph of the NVIDIA Fermi GU-GPU (Source: [12]). 7
1.8 Block diagram of the Hybrid Memory Cube (HMC) (Source:[13]) 9
1.9 An overview of our contributions throughout this dissertation. 11

2.1 Abstract view of the Logarithmic Interconnect (a), and its usage inside a cluster based many-core platform (b). 16
2.2 Block diagrams of the 2D Logarithmic Interconnect (2D-LIN) 18
2.3 Implementation of arbitration methods: (a) PRR, (b) PLRG 20
2.4 Block diagrams of the Centralized 3D Logarithmic Interconnect (C-LIN) 22
2.5 Block diagrams of the Distributed 3D Logarithmic Interconnect (D-LIN) 23
2.6 Structure of 3D stacking and boot-time configuration circuitry (a), delay elements to remove the high-current glitches (b). 24
2.7 Overview of the hierarchical flow for design of C-LIN and D-LIN 25
2.8 Performance comparison between NoC and LIN under different working set sizes. 27
2.9 Physical implementation of the designs: 2-LIN with 2MB SRAM (a), Details of the landing pads on redistribution layer (RDL) in D-LIN (b), Logic Die of D-LIN with Cu-Cu Direct bonding (c), Memory die of D-LIN with details of the TSV Matrix (d), 3D Stacking with 4 stacked memory dies (e). 30
2.10 Comparison of silicon area (mm^2) (a), Maximum achievable frequency (MHz) (b). 31
2.11 Effect of TSVs and their drivers on the critical path of the 3D designs. 31

3.1 Overview of the target architecture and memory map for 3D-NUMA memory IP 36

3.2 Block diagram of the 3D-NUMA memory IP (a) and its configurable parameters (b). 40
3.3 Implementation of the Read-Buffer for $C = 2$, $MOT = 4$. 42
3.4 Schematic view of request-grant FIFOs (a), Join (b), Fork (c) components. 44
3.5 Experimental setup for design and exploration of 3D-NUMA 45
3.6 Schematic view of the 3D-NUMA design to implement 46
3.7 Physical implementation of 3D NUMA in STMicroelectronics CMOS-28nm Low Power Technology. 48
3.8 Average Memory Access Time ($AMAT$) in cycles and its variation (std. dev.) for PARSEC benchmarks, plotted for different number of stacked memory dies (L). Total L2 memory sizes changes from 512 KB to 4 MB proportional to L. 49
3.9 Effect of requested bandwidth on Average Memory Access Time ($AMAT$) (a) and delivered bandwidth (b) (normalized to the ideal case: 64 GB/sec.), where uniform random traffic with packet inter-arrival time of $Random[0, T]$ is applied. 50
3.10 Comparison of the execution time (MCycles) between Scenario 1: 3D-NUMA (a), and Scenario 2: memory banks directly attached to NIs (b), for different PARSEC benchmarks (c). 52
3.11 Comparison of two scenarios for 3D-NUMA: Word-level-interleaving [WLI] (a) and Bank-level-interleaving [BLI] (b); using PARSEC benchmarks (c) (Values of WLI are normalized to BLI) 53
3.12 Comparison of two scenarios for 3D-NUMA under uniform random traffic with packet inter-arrival time of $Random[0, T]$ (Values of WLI are normalized to BLI). 53
3.13 Effect of Maximum Outstanding Transactions (MOT) on execution time ($KCycles$) (a), Average Memory Access Time ($AMAT$), and access time variation, in cycles (b). Effect of MOT on the distribution of Memory Access Time (MAT) in cycles (c), full-bandwidth uniform random traffic is applied. 54
3.14 Total executed cycles and execution time (μS) for different configurations leading to the same total memory size ($S \times L \times C = 2MB$), where uniform random traffic is applied 55
3.15 Power consumption breakdown in the logic die (a), and in memory die MD0 with automatic clock gating (b), comparison of power in the stack of 8 memory dies for four different clocking strategies (c), energy/transaction (nJ) compared in the same experiment (d). In all experiments, uniform random traffic has been applied with a packet inter-arrival time of *Random[0,T]*). 57
3.16 Reduction in switching activity: in a stack of 8 memory die with *PCL* (a), in memory dies with *PCM* (b), total consumed power in the stack of 8 memory dies compared between the clock gating methods (c) . . . 59
3.17 Four configurations for 3D-stacking of 3D-NUMA: Memory-Processor-Interposer (MPI) (a), Memory-Interposer-Processor (MIP) (b), Processor-Interpose-Memory faced Down (PIMD) (c), and Processor-Interposer-Memory faced Up (PIMU) (d) 60

3.18 Cross section of the PIMD configuration 60
3.19 Temperature cool down rate (° K/Seconds) of the PE Die in the four configurations (a), and its transient temperature (° K) increase due to constant power consumption in the PE die [T_{SS}: steady state temperature] (b), Effect of power consumption (W) in the PE die on its temperature (° K)(c), PE die temperature versus the number of stacked memory dies [L] (d). 62
3.20 Final temperature map of the PE die in the four configurations: MIP (a), MPI (b), PIMD (c), PIMU (d), cross-sections of the whole stacks (e,f,g,h). 64
3.21 Manufacturing yield versus TSV failure rate compared between 3D-NUMA with no TSV redundancy, TSV Repair Scheme (Up to 5 Fixes), and the 2D counterpart 67
3.22 Overall manufacturing cost of one PIMD stack compared its 2D counterpart 67

4.1 Overview of the Smart Memory Cube (SMC) 73
4.2 Proposed AXI 4.0 based logarithmic interconnect for SMC 75
4.3 5-layer substitution-permutation network for 22 address bits (S.22.5.05) 79
4.4 Quality of 22-bit substitution-permutation networks vs. number of layers. 79
4.5 Modified 4-layer address scrambler for 22 address bits (S.17.5.04) . . . 80
4.6 Address scrambler for improving sequential read with additional signal to enable or disable scrambling (S.17.5.04.T) 81
4.7 Three additional scrambling alternatives: *S.14.8.04* (a), *S.14.8.04.T* (b), *S.14.8.00* (c). 82
4.8 Schematic view of a Vault Controller 83
4.9 Delivered bandwidth of one vault only (a) and the baseline HMC (b). Zero load latency breakdown for READ (c) and WRITE (d) commands. 87
4.10 Effect of page policy on delivered bandwidth from SMC 87
4.11 Effect of R/W ratio of random traffic on total bandwidth delivered from SMC 88
4.12 Effect of AXI data width (a) and DRAM bus width (b) on delivered bandwidth from SMC 89
4.13 Effect of DRAM t_{CK} (a) and AXI Clk period (b) on bandwidth delivered by SMC 90
4.14 Effect of scaling down the timing parameters of DRAM (a) and number of banks per vault (b) on bandwidth delivered by the SMC 91
4.15 Effect of conventional address remapping (a), and address scrambling (b) on delivered bandwidth of SMC 91
4.16 Effect of changing the step size of synthetic linear traffic on delivered bandwidth and execution time. 93
4.17 Delivered bandwidth compared between different addressing schemes, compared for linear walk (a), and random walk (b). The nodes size of the data structure has been changed from 256 Bytes to 4096 Bytes. . . . 94

4.18 Heat map for the DRAM banks plotted over time, for linear walk on data structure nodes with different node sizes. In the upper plots address mapping is *LOW.INTLV*. In the middle plots the addressing scheme is S.14.8.00, and in the bottom ones S.17.5.04.T is utilized. 95
4.19 Heat map for the DRAM banks plotted over time, for three benchmarks: blackscholes, dense-matrix-add, and streamcluster. In the upper plots address mapping is *LOW.INTLV*, and in the bottom ones address mapping is S.17.5.04. 95
4.20 Effect of Row-major (RM) and Column-Major (CM) matrix traversal methods on delivered bandwidth. 96
4.21 Increase in average (a) and maximum (b) memory access time caused by PIM. Delivered bandwidth to PIM as a function of requested bandwidth on PIM port (c), Drop in main bandwidth caused by interference of PIM (d) 97
4.22 Effect of requested bandwidth by ideal PIM on x264 (a), and effect of buffer size of double-buffering PIM on x264 (b). 98
4.23 Effect of increasing the number of PIM ports on delivered bandwidth (a) to the main links, AMAT of the main links (b), PIM's delivered bandwidth (c), maximum MAT (b) 99

5.1 An overview of the SMCSim Environment for Design Space Exploration of the Smart Memory Cube (SMC) 105
5.2 A sample page to frame mapping highlighting the merged pages (a), The *slice-table* data structure (b), values stored in the *slice-table* for current example (c) 107
5.3 Visualization of the simulated system in SMCSim composed of the host processors, the interconnects, SMC controller and the serial links, the SMC model, and the PIM subsystem. 109
5.4 PIM's software stack 110
5.5 An overview of the accuracy comparison methodology between the high-level SMC model developed in gem5 and the cycle-accurate SMC model. 111
5.6 Traversing the sparse graphs represented using LIL format using two DMA resources 114
5.7 Offloading overheads in execution of the ATF kernel 114
5.8 PIM's speed-up with/without SMC Atomic Commands (left axis), LLC hit-rate associated with the data port of the executing CPU (right axis) 115
5.9 PIM's speed-up versus cache block size of the host 116
5.10 Effect of PIM's TLB size on hit-rate and speed-up 116
5.11 Effect of DMA transfer size on PIM's execution time 117
5.12 Achieved energy efficiency for using PIM versus total stacked memory size (a), power breakdown for execution of the host (b) and PIM (c) . 118
5.13 PIM's energy efficiency versus its clock frequency. 119
5.14 Simulation setup for comparison of PIM with a host-side accelerator with similar capabilities. 119

Chapter 1

Introduction

The memory hierarchy in the modern computer systems plays a crucial role and highly impacts their performance and energy consumption. With the advancements in the processor technology, computational capabilities and transistor count of the processors are increasing rapidly. This has been coupled with a steady growth in the memory density illustrated in Figure 1.1. For these reasons, the amount of bandwidth theoretically available to each transistor has been decreasing exponentially (See Figure 1.2). Also, the emergence of data-intensive workloads such as graph traversal [14] and scientific applications [15], requiring fast transfer of large volumes of data, has put an even higher pressure on the memory system. As a result, an increasing portion of time and energy in computing systems is spent on data movement, especially in off-chip memory accesses [16]. This increasing gap between computation speed and data transfer is known as the "memory wall" problem.

Design of the memory hierarchy has always been challenging due to the incompatibility of the DRAM and the logic process technologies [17]. Recently, the integration of DRAM in logic processes has achieved some partial success, but it has always been plagued by high cost and low memory density issues [18]. Under these circumstances, logic and DRAM are placed on separate dies, and the communication between them has been provided by the high latency and power hungry IO pins. On the other hand, SRAM-based caches and memories can be accommodated in the logic dies, nevertheless, their size is extremely limited and highly impacts the manufacturing yield and cost of the logic process [19].

With the emergence of heterogeneous Three Dimensional (3D) Integration based on through-silicon-vias (TSV), this situation has started to recover in the past years. Because, it is now possible to improve memory access bandwidth and/or latency by

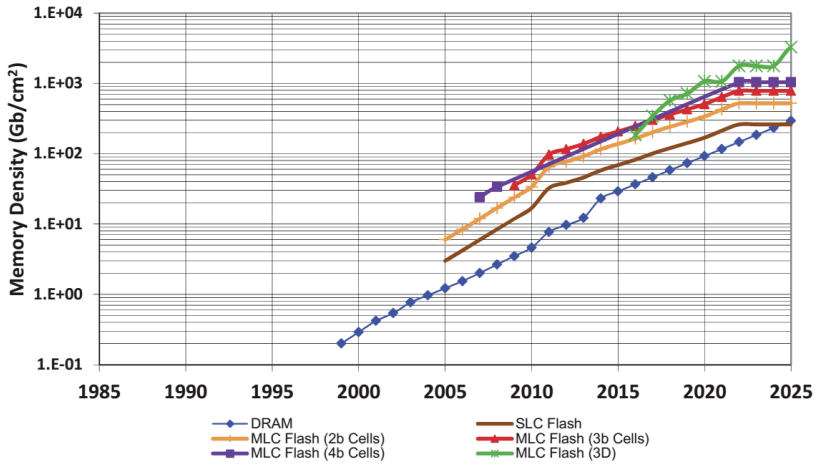


Figure 1.1: Steady growth in the memory density over the years (Source:[8])

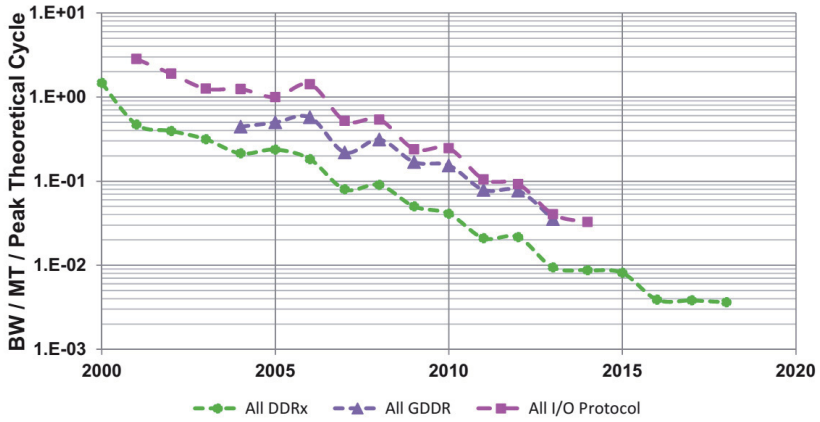


Figure 1.2: Off-chip Bandwidth per Million Transistors at Peak Theoretical Clock Rate (Source:[8])

either stacking memories directly on top of processors as in the WIOMING Wide-IO DRAM shown in Figure 1.3.a, or through abstracted memory interfaces such as Micron's Hybrid Memory Cube (HMC) shown in Figure 1.3.b.

The Three-dimensional (3D) Integration Technology has been explored in academia and industry for over a decade now, and a wide variety of technologies, materials, and processes have been used for research and demonstrations. Several vertical interconnect technologies have been explored, including wire bonding, microbump, contactless (capacitive or inductive), and through-silicon-via (TSV) vertical interconnect [20]. Among them, the TSV approach has gained popularity, due to the high interconnection density. From the TSV technologies, MITLL [20] and Tezzaron TSV [21] offer high density (over

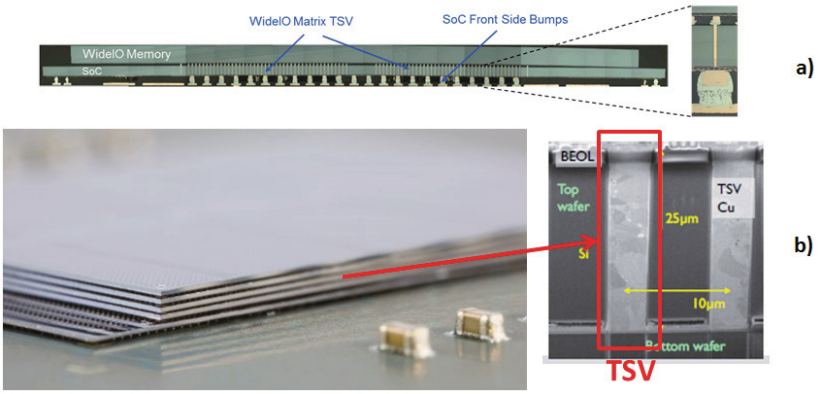


Figure 1.3: Cross-section of the WIOMING Wide-IO DRAM (Source: [9]) (a) and the HMC's die photograph (Source: [10]) (b).

15000 via/mm^2) and low resistance (<0.5 Ohm) and capacitance (<2 fF), however, their number of stacked layers is limited to 3 and 2 respectively, and they are used in technologies larger than 90 nm and in low-volume production. The current state of the art [22][23][24] in high-volume production-ready TSV technology uses more conservative spacing (< 10000 via/mm^2) and physical and electrical interfaces (≃30 fF). TSV technology was brought to commercial maturity by memory manufacturers (DRAM and Flash) to build "memory-cubes" made of vertically stacked thinned memory dies which achieve higher capacity in packages with smaller footprint and power compared to traditional multi-chip modules. In fact, one of the biggest drivers for high-volume adoption of the 3D-Integration technology has been 3D memory stacking with three main classes of: 3D DRAM main memories, 3D caches, and 3D Scratchpad Memories (SPM) [25].

3D stacked DRAM memories offer larger capacity and higher bandwidth in comparison with traditional DDR devices. The most outstanding examples of 3D stacked DRAMs are the Hybrid Memory Cube [13], the High Bandwidth Memory (HBM) [26], and the Exascale Memory [27]. 3D stacking of caches, which is an approach still at advanced R&D stage, has been intensively investigated, as well. 3D stacked caches with wide I/O interfaces [28][29] and 3D stacked non-uniform cache architectures (NUCA) [30][25][31] are just a few samples in this context. In contrast with caches, SPMs are visible in the System-on-Chip (SoC) memory map, and are suitable for data-structures which are not well-managed through caches. L1 SPMs offer very low latency access to a cluster of tightly-coupled processors. L2 scratchpad memories exhibit lower sensitivity to memory access latency and its variations. This makes them another interesting

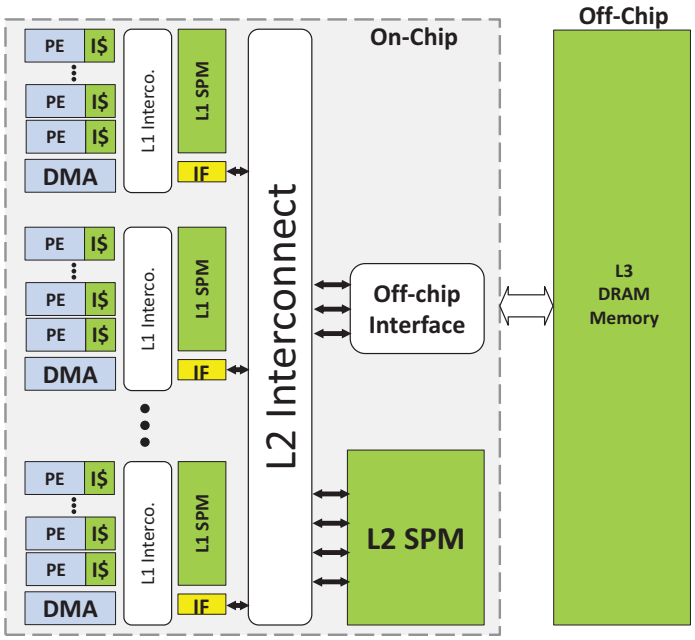


Figure 1.4: An overview of a typical memory hierarchy in the cluster-based many-core platforms.

candidate for going towards the third dimension. In addition, SPMs can coexist with the caches and most application processors and almost all mobile SoCs feature on-chip scratchpad memories shared by multiple processing elements. Snapdragon 800 Processors by Qualcomm [32], Exynos 5 by Samsung [33], and Keystone II by Texas Instruments, with private/shared caches and scratchpad memories are great examples in this context. We should clarify that by L1 we mean an in-cluster memory which should be accessed without stalling the pipeline of the processors. L2 is an out-of-cluster memory with a typical latency of on-chip inter-cluster access, i.e. a few tens of cycles. And lastly, main (L3) memory is typically off chip, and characterized with a latency of hundreds of cycles (See Figure 1.4)

It has been predicted that 3D TSV chip market will grow more than 10 times faster than the global semiconductor industry [34]. Also, wafer foundries such as Samsung and TSMC have been developing vertical integration offerings to meet with the demand from leading fab-less companies such as Qualcomm, Broadcom, Marvell, nVidia and Apple, along with fab-lite IC companies such as TI, STMicroelectronics (STM) and NEC/Renesas [34]. Nevertheless, the time for adoption of 3D Integration for mass production keeps shifting out into the future. Several technical challenges and infrastructure issues are delaying high volume manufacturing of TSV technology for 3D ICs. Until these issues can be resolved, alternative packages will continue to be used [35].

On the other hand, advanced packaging technologies provide new opportunities for heterogeneous integration, power delivery, cost optimization, and thermal management. Stacked Chip Scale Packaging (SCSP) of Amkor Technology [36] is one such example which provides several different 2.5D/3D options for integration of heterogeneous dies in a package. Among other packaging technologies, Dual DRAM Package (DDP), Dual Face Down (DFD), and Quad Face Down (QFD) [37][38] with the main target of DRAMs provide complex forms of wire-bonding which may be adopted even for other levels in the memory hierarchy. Technologies such as TSV Silicon Interposer (TSI) [39][40] and wafer reconstitution [41] provide even more flexibility in hybrid 2.5D/3D stacking. TSIs enable stacking of different dies on both sides to achieve a better utilization of space and to facilitate heat transfer of high power chips. Wafer reconstitution provides electrical connections from the chip pads to the interconnects by means of an artificial wafer. Redistributed Chip Packaging (RCP) [41] developed by Freescale Semiconductor offers scalable chip-scale packaging and multi-die heterogeneous integration. In addition, Package on Package (PoP) stacking is supported in RCP by means of Through-Package Vias.

Another active trend in the computing domain has been parallel processing where a large number of simple cores are integrated on the same die. The ever increasing focus on the energy-efficient architectures (as shown in Figure 1.5) and slowdown in the improvement in the clock frequency (See Figure 1.6) have been the two main drivers for moving from single processing cores to multi-core and multi-cluster platforms. GP-GPUs such as NVIDIA Fermi [42], HyperCore [43], and ST-Microelectronics Platform 2012/STHORM [44] are the most visible examples in this trend. All of these architectures follow cluster-based many-core designs with a limited number of processors (up to 32) in each cluster. Several levels of scratchpad-memories and caches along multi-channel DRAM memories form their memory hierarchy and serve for different range of requirements from the processing clusters. An overview of the typical memory hierarchy in such platforms is illustrated in Figure 1.4, and the die photograph of the NVIDIA Fermi GP-GPU is shown in Figure 1.7.

Researchers in the early nineties, tried to tackle the "memory wall problem" in a completely different way [45], by looking into the possibility to migrate some part of computation closer to the memory systems. Computational RAM [45] using SRAMs or DRAMs coupled with processing elements close to the sense amplifiers, and Intelligent-RAM (IRAM) [46] to fill the gap between DRAM and processors, are just two examples of the efforts in this area. It was shown by Patterson *et. al.* [46] that in memory

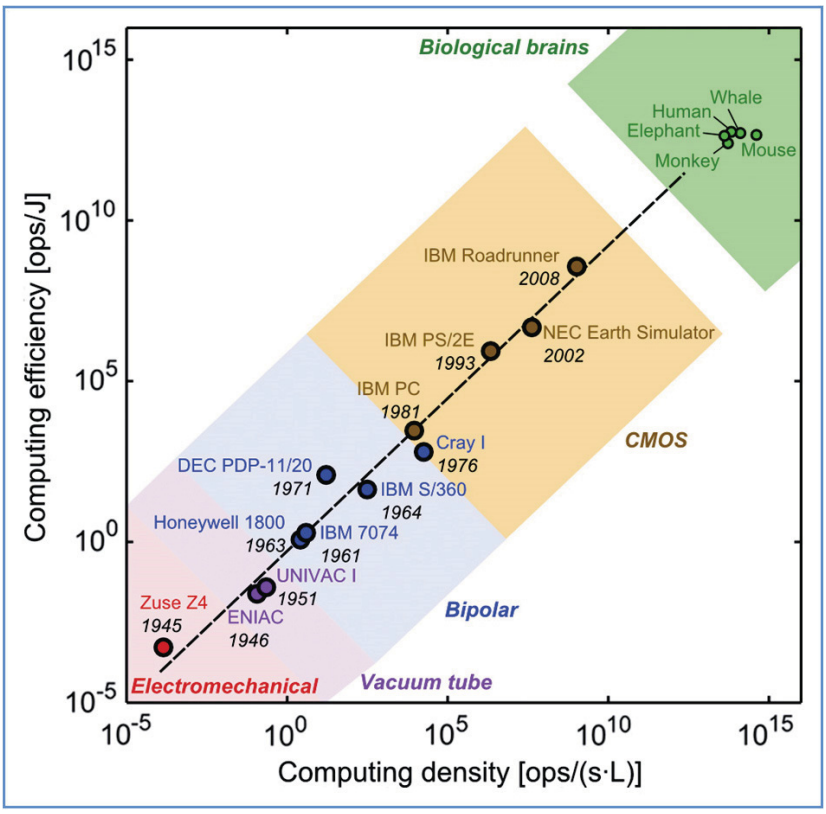


Figure 1.5: A comparison of the computing efficiency among the biological brains and different computation technologies (Source:[11]).

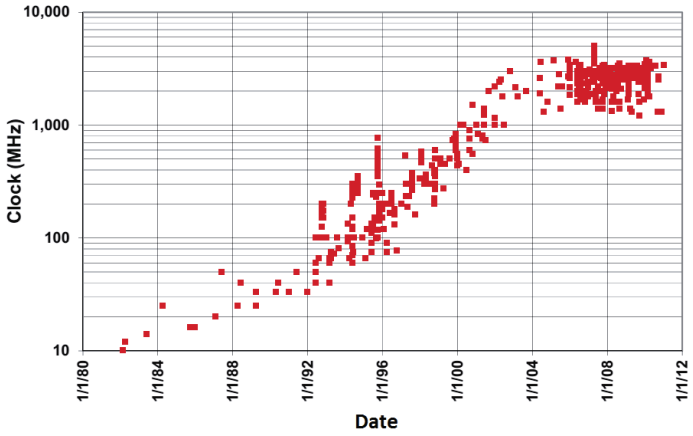


Figure 1.6: Historical Clock-rates (Source:[8]).

processing can lead to a memory bandwidth and energy efficiency improvement of 50X∼100X and 2X respectively, along with a latency reduction of about 2X. Unfortunately, the "processing-in-memory" research efforts in the late nineties and the first decade of the new millennium (See [45][46][17] for samples) did not lead to successful

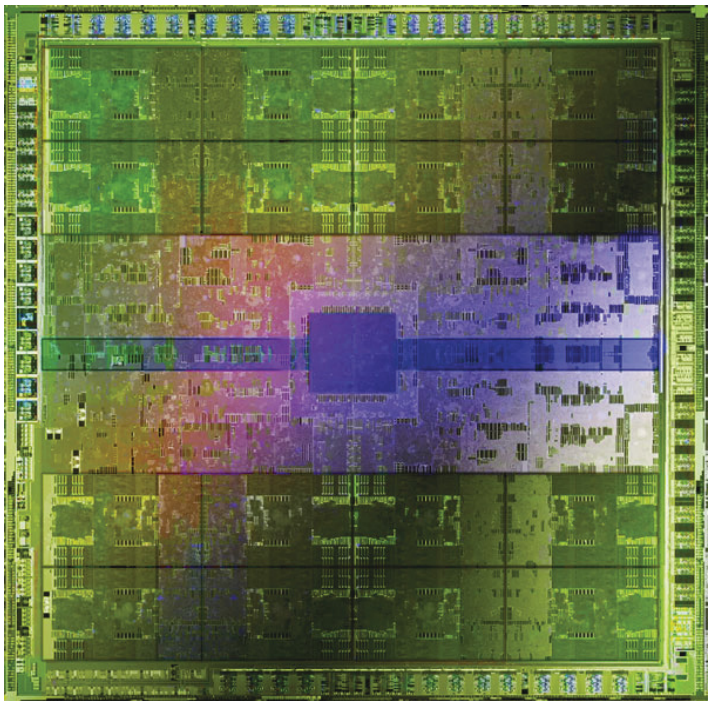


Figure 1.7: Die photograph of the NVIDIA Fermi GU-GPU (Source: [12]).

industrial platforms and products. The main reason for this lack of success was that all these works were assuming that significant amount of logic resources, needed for having processing elements close to the memory arrays, could be integrated on DRAM dies (or vice versa). This could not be achieved economically given the restrictions of DRAM processes (e.g., limited number of metal levels, slow transistors).

With the appearance of the heterogeneous 3D integration of logic and memory dies based on through-silicon-vias (TSV), this situation has started to change, and the interest in near-memory computation has been renewed [47]. Loh *et. al.* [48] study a fixed function Processor-in-Memory (PIM) in a 3D-stacked context as a starting point. Farmahini *et. al.* [49] demonstrate a Coarse Grain Reconfigurable Accelerator (CGRA) located on a separate die and connected to the DRAM dies through Through Silicon Vias (TSVs). Islam *et. al.* [15] present how to form a PIM cluster with 64 Cortex-A5 processors and two levels of caches. Active Memory Cube (AMC) [50] extends the logic layer of the HMC with clusters of vector processors without caches. In [51] PIM is comprised of CPUs and GPUs. [52] augments the logic die with a cluster of 16 light-weight general purpose cores with 2 levels of caches. Tesseract [53] features a network of memory cubes each accommodating a cluster of in-order cores with L1 caches.

Near memory computation can now exploit two main opportunities provided by 3D integration: (1) vicinity to the main storage resulting in reduced memory access latency and energy, (2) higher bandwidth provided by Through Silicon Vias (TSVs) in comparison with the interface to the host, limited to the pins. The last missing piece came in place when an industrial consortium backed by several major semiconductor companies introduced HMC. [13] In the HMC, a memory cube is stacked on top of a logic die (See Figure 1.8). The logic die at the bottom of the hybrid stack provides an advanced communication interface between the memory cube on top and the rest of the computing system on the board. Fast serial IO transceivers for off-chip communication, and on-chip controllers and interconnects for multiplexing the vertically stacked memory partitions (called "vaults"), are all implemented in it. The communication mechanism follows a packet-based protocol implementing different networking layers (physical, link, and transaction layers). It also supports advanced flow-control and congestion control mechanisms [13]. This form of abstraction hides the details of the DRAM control from the host and provides a flexible and standard communication infrastructure to be used by different host platforms. Also, apart from the performance and energy benefits [54], this mechanism allows for supporting higher-level commands (e.g. atomic operations [13]), in addition to the conventional read and write commands. Because of this flexibility and abstraction, we believe that HMC is the best target for hosting processor in memories.

To summarize, in the past years, the focus on high-performance computing along with energy efficient architectures have lead to an ever increasing interesting in parallel computing systems, increasing the pressure on the memory hierarchy. 3D integration has created two optimization opportunities to hopefully solve this "memory wall" problem:

- Improving memory access bandwidth and/or latency by either stacking memories directly on top of processors, or through abstracted memory interfaces such as Micron's Hybrid Memory Cube (HMC).

- Revisiting near memory computation due to the cost-effective integration of logic and memory in 3D stacks, by placing logic and memory close to each other, but in their own optimized processes, and provide high bandwidth connection between them by means of TSVs.

In this research, we exploit these two opportunities and study the effectiveness of the 3D integration technology in the different layers of the memory hierarchy in cluster-

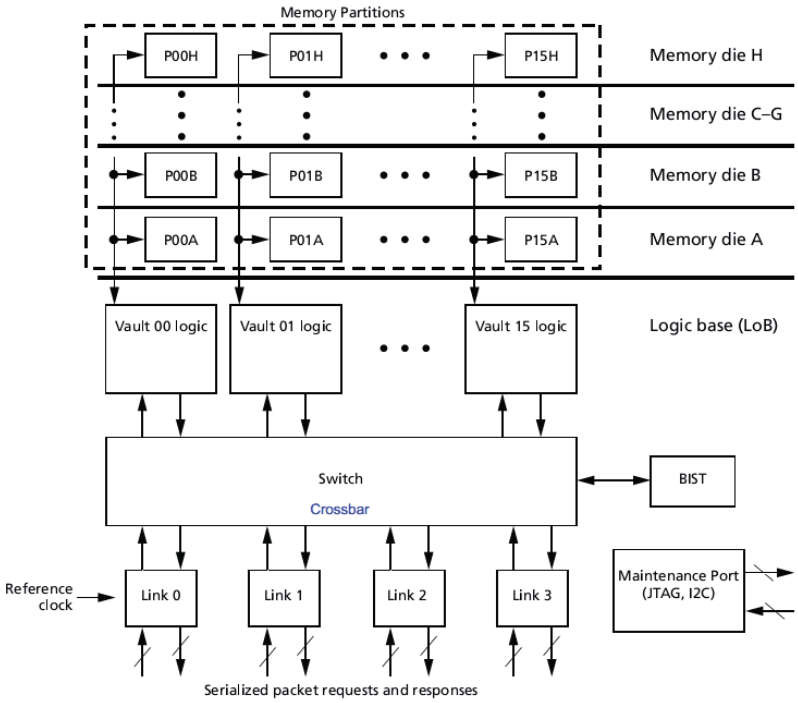


Figure 1.8: Block diagram of the Hybrid Memory Cube (HMC) (Source:[13])

based many-core platforms ranging from intra-cluster L1 to inter-cluster L2 scratchpad memories (SPMs), as well as the main memory. We investigate the provided optimization opportunities in terms of performance, power consumption, and silicon area. In addition, by moving a part of the computation to where data resides we demonstrate further optimization opportunities. Apart from improvements in performance, energy, and area, we also keep an eye on scalability, modularity, and manufacturability issues, and try to propose flexible solutions reducing time-to-market and manufacturing cost and effort. We demonstrate that with the current TSV technologies, moving towards the third dimension inside the processing clusters can only be beneficial in terms of modularity, flexibility, and manufacturing cost. While, to achieve significant performance improvements the next levels of the memory hierarchy should be explored.

Our proposed solutions range from circuit-level optimizations to minimize high-current glitches in the 3D stacked interfaces to avoid degradation of the chip's life and performance; to system level clock gating and power reduction strategies; and software level coordination of computation for the sake of energy and performance. Our obtained results are backed up by physical implementation of cycle-accurate models using industrial technology libraries, as well as, calibrated full-system simulation envi-

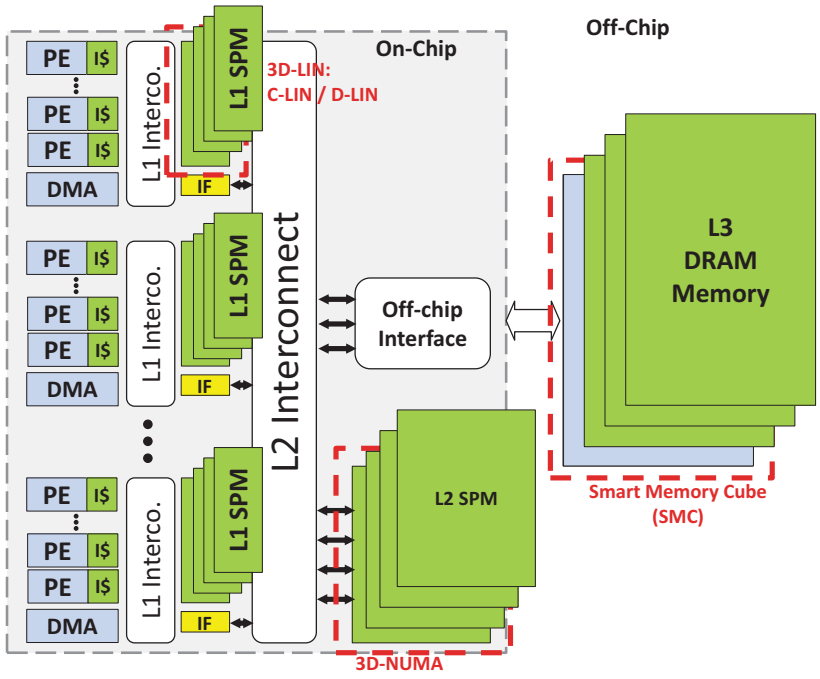


Figure 1.9: An overview of our contributions throughout this dissertation.

ronments. We have used different industrial and academic platforms in synergy with each other to achieve accurate and realistic conclusions. Here we present the outline of this dissertation.

1.1 Organization of Dissertation

This dissertation is structured as follows: In chapter 2 we investigate the benefits of moving towards the third dimension in a processor-to-L1-memory context (See Figure 1.9). We focus on a single processing cluster sharing a tightly coupled multi-banked L1 scratchpad memory, and propose two 3D network architectures: Centralized Logarithmic Interconnect (C-LIN) and Distributed Logarithmic Interconnect (D-LIN) to provide low-latency L1 memory access through 3D interfaces. We propose a modular stacking strategy which allows stacking of multiple identical memory dies. Our designs have been implemented using two bonding techniques with consideration of the electrostatic discharge (ESD) protection circuits. In subsection 2.6.2, we point out the obstacles which have prevented 3D integration in the L1 context (L1 caches, L1 SPMs, and processors) from being successful in the industry, despite the general belief that TSVs can greatly reduce wire length and improve clock frequency.

In chapter 3, we extend our studies to the L2 memory context and focus on L2 scratchpad memories located outside the processing clusters (Figure 1.9) because of their requirement for large sizes and lower sensitivity to memory access latency and variations. Also, because most application processors and almost all mobile SoCs feature a large on-chip L2 memory which is shared by multiple cores. We will present 3D-NUMA, a 3D L2 memory IP, which can be attached to a cluster based multi-core platform through its NoC Interfaces, and offer high-bandwidth memory access with low average latency. Our proposed IP allows modular stacking of multiple memory dies with identical layouts using a single mask set, supports multiple in-flight transactions, and achieves high clock frequency thanks to its highly pipelined nature. Several experiments are performed to evaluate 3D-NUMA in terms of performance, power consumption, thermal behavior, and manufacturing yield and cost.

In chapter 4 and chapter 5, we will move towards the last level in the memory hierarchy, and study the benefits provided by 3D integration, there (Figure 1.9). We choose the Hybrid Memory Cube (HMC) as the most outstanding industrial example, and introduce the concept of the "Smart Memory Cube (SMC)": a fully backward compatible and modular extension to it, supporting near memory computation. In chapter 4 we focus on the architectural implications and the required infrastructure inside HMC to support this feature. We propose a high performance and extensible AXI-4.0 based Logic-Base (LoB) interconnect, carefully designed to provide high bandwidth to the external serial links, as well as plenty of extra bandwidth to any generic and AXI-compliant PIM device attached to its extension ports. We also implement a novel address scrambling mechanism for reducing vault/bank conflicts and robust operation in presence of pathological traffic patterns. Cycle accurate (CA) models for the SMC interconnect and its interfaces are developed, and their parameters are tuned based on the available data from the literature on HMC.

Lastly, in chapter 5 we explore the design of a PIM architecture for the SMC introduced in chapter 4. A full-system simulation environment called SMCSim has been developed and verified against the Cycle-Accurate (CA) model described in chapter 4. SMCSim models the complete hardware and software stack ranging from high-level user application to low-level firmware and hardware layers. It takes into account the offloading and dynamic overheads caused by the operating system, cache coherence, and memory management, as well. We propose an optimized memory virtualization scheme for zero-copy data sharing between host and PIM; enhance the PIM architecture's capabilities by the aid from atomic in-memory operations; and improve its memory

access by means of a flexible Direct Memory Access (DMA) engine.

The main contributions of this dissertation can be summarized as follows:

- **Design and physical implementation of two 3D network architectures for modular and flexible stacking of L1 scratchpad memories**
  Our proposed designs offer modularity and better scalability in comparison with their 2D counterparts. However, in terms of delay, they are so competitive yet. We observed that since the current TSV technologies are not yet so competitive with on-chip wires, for small sized L1 memories 3D integration does not seem to be beneficial in terms of performance.

- **Design and physical implementation of a 3D L2 SPM IP called 3D-NUMA offering high-bandwidth memory access with low average latency**
  We demonstrate that addition of 3D-NUMA to a multi-cluster system can lead to an average performance boost of 34%. Moreover, through several experiments we show that 3D-NUMA is energy and power efficient, temperature friendly, and has unique features suitable for low cost manufacturing.

- **Introduction of the Smart Memory Cube (SMC), a fully backward compatible extension to HMC, supporting near memory computation on the LoB die, and Design of a high performance AXI-compatible LoB interconnect for it**
  Our designed interconnect easily meet the demands of current and future projections of HMC (Up to 205 GB/s READ bandwidth with 4 serial links and 32 memory vaults). The interference between the PIM traffic and the main links is found to be negligible when PIM has up to 2 ports requesting up to 64 GB/s. It is further shown that low-interleaved addressing is not reliable enough for an abstracted memory such as HMC. Fat data structures with power-of-two node sizes are particularly identified as unfavorable patterns for low-interleaving. A more robust address scrambling mechanism is proposed to effectively reduce bank/vault conflicts. Logic synthesis confirms that our proposed models are implementable and effective in terms of power, area, timing (power consumption less than 5mW up to 1 GHz and area less than $0.4mm^2$).

- **Design of a processor in memory architecture for SMC featuring an optimized memory virtualization scheme for zero-copy data sharing**

between host and PIM

It is shown that even in a case where the only benefit of using PIM is latency reduction, up to 2X performance improvement in comparison with the host SoC, and around 1.5X against a similar host-side accelerator is still achievable. By scaling down the voltage and frequency of the proposed PIM it is possible to reduce energy by about 70% and 55% in comparison with the host and the accelerator, respectively.

A summary of the obtained results and conclusions is given at the end of each chapter, while chapter 6 gives a general conclusion to the dissertation. References [1, 2, 3, 4, 5, 6, 7] have been published throughout the work on this thesis.

Chapter 2

3D Stacking of L1 Scratchpad Memories

In this chapter we focus on shared tightly coupled data memories (TCDMs), as they are the key architectural elements for building multi-core clusters in programmable accelerators and embedded systems, and they provide a convenient shared memory abstraction while avoiding cache coherence overheads. We use 3D integration to increase modularity and scalability of the L1 TCDMs, and study the effects of going vertical on their performance and silicon area. We propose two 3D network architectures: C-LIN and D-LIN, which allow modular stacking of multiple L1 memory dies over a multi-core cluster with a limited number of processing elements.

2.1 Motivations and Challenges

The increasing focus on energy-efficient architectures coupled with a slowdown in clock speed improvement has created a growing interest in parallel computing where a large number of simple cores are integrated onto the same die. GP-GPUs such as NVIDIA Fermi [42], HyperCore [43], and ST-Microelectronics Platform 2012/STHORM [44] are the most visible examples in this trend. All of these architectures follow cluster-based many-core designs with a limited number of processors (up to 32) in each cluster sharing tightly-coupled L1 data memories (TCDM), a.k.a scratchpad memories (SPM). TCDMs are useful for the data-structures which are not well-managed through the caches. Together with Direct Memory Access (DMA) engines they can lead to more efficient implementations with more predictable behaviors [55].

Network-on-chips (NoC) designs have been advocated as an alternative to bus-based

architectures; thanks to their scalability which makes them suitable for inter-cluster communication and in L2 and upper levels of the memory hierarchy. However, their high average latency and latency variability, as well as increased design complexity to guarantee correctness and fairness (e.g., avoiding deadlock, livelock, starvation) [56] brings their usefulness in processor-to-L1-memory context under question.

On the other hand, crossbar-based interconnects, like the one in IBM BlueGene/Q [57], can provide a uniform and ultra-low memory access latency within a cluster, which is not achievable in multi-stage NoC systems. The design of crossbar networks for high-performance usually relies on custom circuit design techniques such as pass transistors and low-swing drivers (e.g. [58]). Full-custom approaches are not suitable for architectures featuring soft cores and third-party Intellectual Property (IP) blocks, and their reusability is limited across technology nodes. Therefore, processor-to-L1-memory interconnects provided as a parametric synthesizable IP are highly desirable in this context.

In this chapter, we take advantage of the 3D integration technology to increase the shared L1 memory size in a processing cluster in a modular fashion, i.e. stacking memory dies on top of a logic die, without the need to re-spin silicon (as it would be needed for traditional 2D technology). We focus on a single cluster with a typical size (16 processing elements) sharing a tightly coupled multi-banked L1 memory (See Figure 2.1.b), and propose two 3D network architectures, C-LIN and D-LIN (designed in synthesizable RTL) which can be configured based on user specifications and technology constraints to provide low-latency memory access. Our modular stacking strategy allows system integrators to stack multiple memory dies and create arbitrary L1 memory sizes through different height stacks with identical dies, without the need for different masks for dies at different levels in the stack. The designs have been implemented in STMicroelectronics (STM) CMOS-28nm Low Power technology library, using two bonding techniques with consideration of the electrostatic discharge (ESD) protection circuits.

Related research efforts are presented in section 2.2. 2D-LIN and its 3D extensions are described in section 2.3 and section 2.4. Issues related to the 3D Integration are discussed in section 2.5, and finally, experimental results and a summary of conclusions are presented in sections section 2.6 and section 2.7.

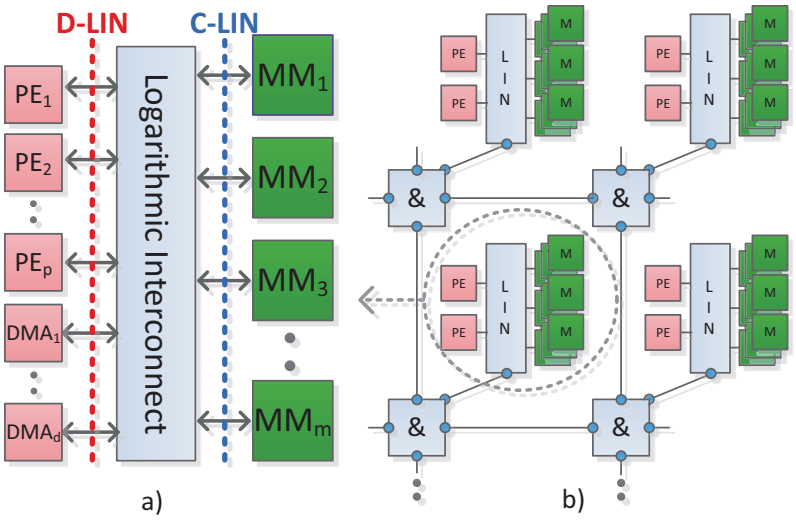


Figure 2.1: Abstract view of the Logarithmic Interconnect (a), and its usage inside a cluster based many-core platform (b).

2.2 Related Works

Performance limitations of the interconnection networks have led to a renewed interest in interconnect research and a transition from traditional bus-based systems to more sophisticated topologies, including mesh network-on-chips [59], hierarchical bus models [60], flattened butterfly on-chip networks [61], and crossbars [62], [19], and [63]. The ability of crossbars to provide uniform access latency makes them an appealing option in processor-to-L1 memory interface for limited-cardinality clusters (16 PEs, typically) illustrated in Figure 2.1.b. Because predictable access latencies allow for quality-of-service guarantees and ease of programming. Custom designed crossbar-switches can provide very high bandwidths (e.g., 1Tbit/s in [58]), however, lack of configurability and their incompatibility with standard technology libraries provided by silicon foundries make them unsuitable to SoC Design. Crossbar networks for tightly coupled shared memories have been used in the HyperCore Architecture [43], which contains a shared on-chip memory accessed through a series of combinational switches; and in [19] and [62] using Mesh of Trees (MoT) and Swizzle Switch Networks (SSN), respectively.

3D stacking of scratchpad memories to replace fast on-chip SRAMs has been studied in [64] and [65]. In [64], the authors proposed a configurable memory layer that consists of many uniform memory elements connected directly to processors. In [65], a prototype of 3D stacked TCDM has been published, which is a two-layer 3D IC, with logic die consisting of 64 general-purpose processor cores running at 277MHz and connected through a mesh NoC, and the memory die with 256KB of SRAM. While these works

have simply focused on the use of private memory banks, our work proposes a solution for sharing L1 memory, and shows that NoC solutions are not suitable in this context.

Finally, 3D extension of low-latency crossbars for shared L1 clusters has been investigated in [63] and [62], while [66] uses time-division multiplexing buses for this purpose. The key difference between our proposed approach and these works is modularity, which allows stacking of several memory dies on a logic die without the need for new masks for each stacked die. Moreover, our solutions offer better scalability compared to [63] (More in depth discussion is performed in section 2.5 and section 2.6). Also, physical synthesis on realistic 3D floorplans make our obtained results more accurate.

2.3 2D Logarithmic Interconnect

The basic 2D Logarithmic Interconnect (2D-LIN) is a low-latency and flexible crossbar that connects multiple processing elements (PEs) to multiple SRAM memory modules (MMs) (See Figure 2.1.a). The IP is optimized to provide fast arbitration and single cycle access to TCDM banks, as well as, synchronization mechanisms for inter-process communication. It is built following the Mesh Of Trees approach, where the network is created combining binary trees [63]. Each tree provides a unique combinational path between the PEs and one memory module, and vice versa. Therefore, the request and the response paths are decoupled in 2D-LIN to maintain non-blocking communication (See Figure 2.2).

The key property of this soft IP is the reconfigurability. User has control on the following parameters: number of PE channels (p), number of direct-memory-access (DMA) channels (d), number of memory modules (m), size of each memory module in kilobytes (s), and width of data bus (w). Furthermore, bank/word level inter-leaving are both supported, and arbitration can be performed using either Pseudo-LRG or Pseudo-Round-Robin methods, which are modified versions of Least-recently-granted (LRG) and round-robin (RR) to become suitable for implementation inside binary trees. The fraction m/p is defined as *BankingFactor*. When this parameter is less than or equal to one, there will be a high number of collisions between PEs while accessing memory banks, and performance drops severely. Simulation results with different traffic patterns show that a banking factor of 2 offers over 94% of the performance of the ideal case where no collision exists.

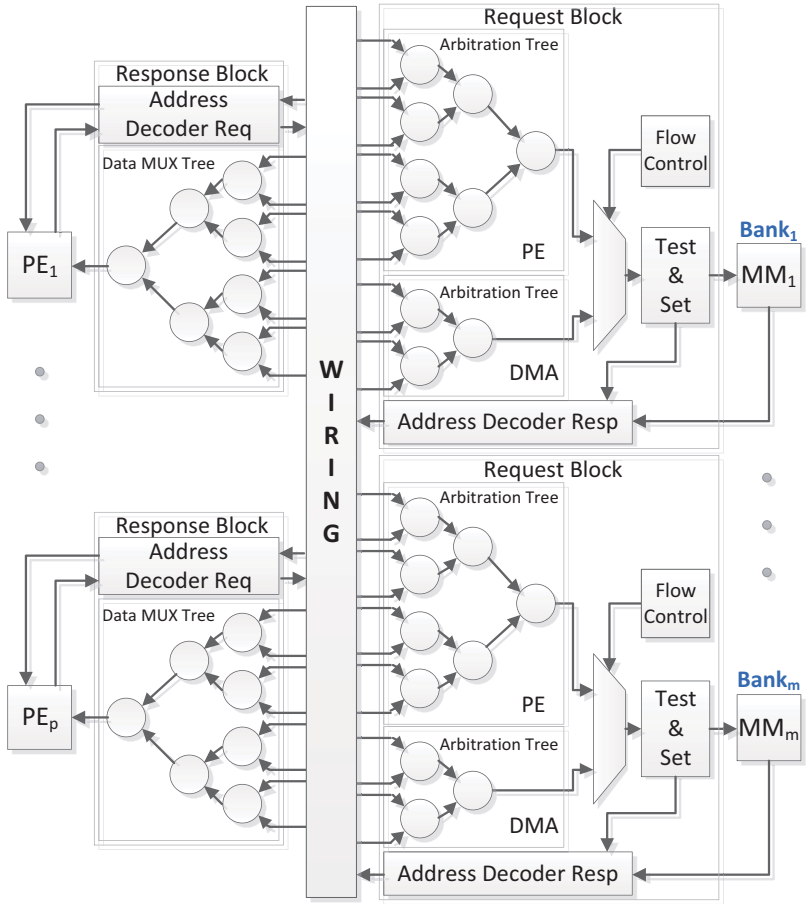


Figure 2.2: Block diagrams of the 2D Logarithmic Interconnect (2D-LIN)

2.3.1 Network Protocol

Each clock cycle, all the requests made from masters (PEs and DMAs) are propagated through the request blocks. Collisions due to multiple requests directed to the same memory bank are avoided by the arbitration performed in each node. PEs losing the arbitration are stalled, and the winners conclude the transfer in a single clock cycle in case of a store, while, in case of a load, the read data is returned the next clock cycle. This architecture implements atomic test-and-set operation as well. It should be noted that, apart from the topology, also the flow control mechanism presented in this work differs from [19] in the way that it operates only at one edge of clock, therefore the clock period can be reduced further, and even though in this method read operations takes two cycles to complete, pipelining of reads allows an average performance of one read per cycle.

Table 2.1: Comparison of different arbitration methods (p=16)

Arbitration	Complexity	LRG-Wait	AMAT	Bias
LRG	-	0	9.85	1.6
RR	-	$p-1$	10.13	2.1
PRR	$log(p)$	$p-1$	9.8	10.3
PLRG	p	$log(p)$	9.63	1.9
SS	p^2	0	9.85	1.6

2.3.2 Request Block

Request block is in charge of collecting all the PEs' requests directed to a specific memory module (see Figure 2.2). This block is built out of a single binary tree with each node being an arbitration primitive. Combining several binary trees, the network can support both generic number of ports and different priorities. Therefore, high priority channels for processors and low priority channel for eventual peripherals are implemented. Arbitration between requests can be implemented using different algorithms. Round-Robin (RR) and Least-Recently-Granted (LRG) are two widely used fair arbitration algorithms; yet, they are not suitable for implementation in a binary-tree. We modified them as Pseudo-Round-Robin (PRR) and Pseudo-LRG (PLRG) to be used inside the logarithmic interconnect. Table 2.1 compares the modified algorithms with RR, LRG, and a good approximation of LRG called Swizzle-Switch Arbitration (SS) [62] which keeps and updates a table of priorities. Results have been gathered from a high-level simulator under different traffic traces (explained in subsection 2.6.1). In this table *AMAT* stands for normalized Average Memory Access Time in Cycles; *LRG-Wait* represents the longest time in cycles which a request may wait in the LRG position before being granted; and *Bias* shows the amount of unfairness between input requests in every one thousand simulation cycles, calculated as in (2.1).

$$Bias = \frac{1000}{Cycles} \times \sum_{i=0}^{p} |grants[i] - 1/p \sum_{j=0}^{p} grants[j]| \tag{2.1}$$

Simulation results show that, on average, all presented algorithms perform similarly and even their unfairness is negligible with respect to the Ideal LRG. Nevertheless, the RR and PRR suffer from the fact that they disregard the request in LRG position and in the worst case that request may have to wait for $(p-1)$ cycles. While, this worst-case latency is improved to only $log(p)$ cycles in the PLRG algorithm. Moreover, PLRG can be implemented with $o(p)$ gates and memory elements, whereas Ideal LRG needs a

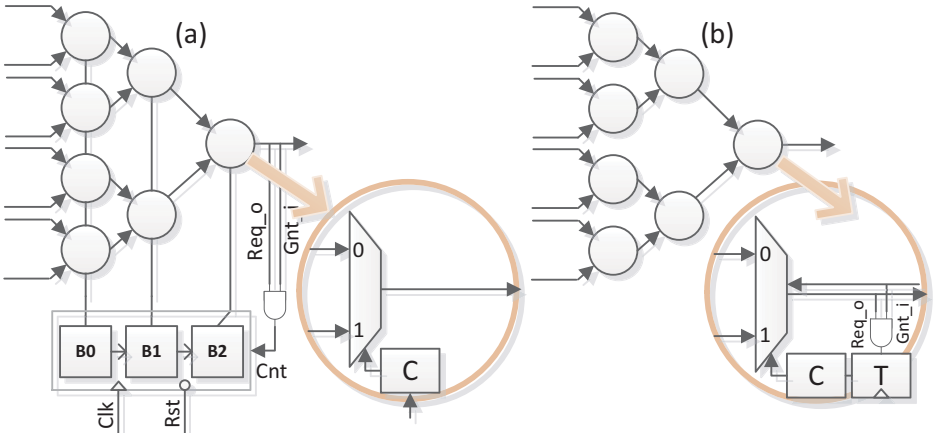


Figure 2.3: Implementation of arbitration methods: (a) PRR, (b) PLRG

stack and SS needs $o(p^2)$ registers and logic. Figure 2.3 depicts the implementation of RR and PLRG inside the logarithmic interconnect. As can be seen, in PRR a simple counter changes the priority of channels every time every time a request is granted. While, in PLRG, each binary arbiter contains a toggle flip-flop, which switches to the unacknowledged input every time a grant is received.

2.3.3 Response Block

Response block is in charge of collecting all the responses from memory modules which are directed to a specific PE (see Figure 2.2). It can be considered as a specular version of the request block. However, since the response network is only used for read operations and the read latency is deterministic (1 cycle), no response collisions are possible, and the response path can be simplified by replacing the arbiters with simpler multiplexers.

2.4 3D Logarithmic Interconnect

Centralized 3D-Logarithmic-Interconnect (C-LIN) and Distributed 3D-Logarithmic-Interconnect (D-LIN) are two extensions of the 2D-LIN to be integrated in a 3D-stacked Chip multiprocessor. These topologies provide more flexibility in comparison to 2D-LIN by automatically splitting the design into one logic layer and several memory layers and stacking them over each other. Both networks have been designed based on the assumption that memory layers with identical layouts will be stacked over

each other, forming vertical memory cones with all their parameters automatically configured during the boot procedure. This allows for reduction in the chip cost and design effort, and adds design flexibility. To allow stacking of identical memory dies, all components on different memory layers share the input data and control signals from the logic die, and tri-state data buses for their responses, as well. Lastly, both C-LIN and D-LIN support configurable parameters of 2D-LIN, plus a parameter l which represents the maximum number of stackable memory layers, though the number of stacked memories can be chosen freely after the 3D chip assembly step. It should be noted that, in the 3D network presented in [63], interconnects are replicated completely in each memory layer, and a copy of the arbitration circuits for each layer is placed on the logic layer. As a result, addition of new memory layers, increases *BankingFactor*, and therefore the size of the logic-die grows. Whereas, in our designs, addition of new memory layers does not affect *BankingFactor* and only adds to the capacity of the existing banks. This makes our solutions more scalable (The comparison between obtained results is performed in section 2.6).

2.4.1 Centralized 3D Logarithmic Interconnect

C-LIN is the simplest extension to 2D-LIN. As illustrated in Figure 2.1.a, the 2D design is cut at the memory interface, therefore, PEs and the interconnection network are placed on the logic layer, while memory modules along with small layer decoding logics are placed on memory layers (See Figure 2.4). One benefit of this architecture is that logic and memory elements are completely separated, therefore, different technologies and optimizations may be utilized for design of the logic and memory dies. In addition, memory layers in C-LIN can be designed as simple, small, and inexpensive as possible (please refer to subsection 2.6.2 and Figure 2.9 for the physical implementation results). The network operation of C-LIN is similar to 2D-LIN with the difference that after the arbitration in the logic layer, the winner request will be sent to memory layers through the TSVs and request address will be matched with *LayerID* (a number which uniquely identifies each memory layer) in the layer decoding logics of each layer. Therefore, the address space is divided among the memory dies, and for each memory bank only one die will be active at a time. This helps maintain the combinational nature of the logarithmic interconnect and avoids insertion of buffers and FIFOs between layers.

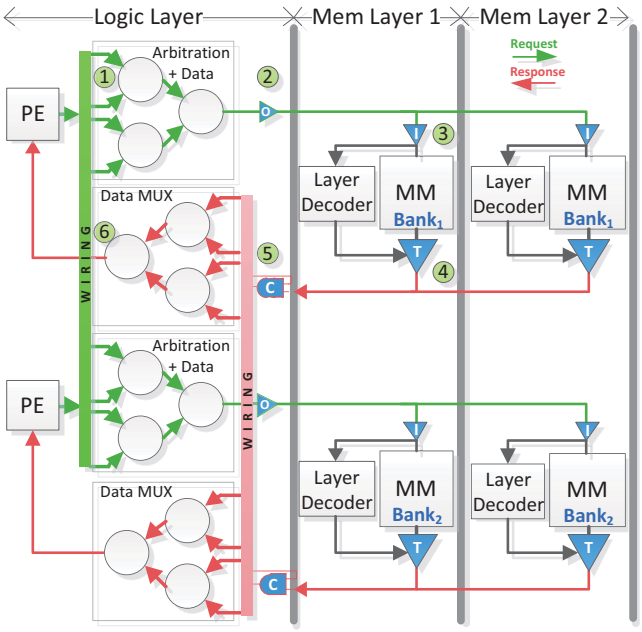


Figure 2.4: Block diagrams of the Centralized 3D Logarithmic Interconnect (C-LIN)

2.4.2 Distributed 3D Logarithmic Interconnect

In the other alternative, D-LIN, 2D design is cut at the PE interface (See Figure 2.1.a), therefore, interconnect is distributed among the layers as illustrated in Figure 2.5. Similar to C-LIN, flow-control is performed in the logic layer, whereas, after arbitration in the logic layer, filtered requests will be propagated to memory layers and in the target memory layer knowing that all the collisions have been already resolved, simple multiplexer trees send data into MMs. Response networks also act similar to C-LIN, with the difference that they are located in memory layers. In both C-LIN and D-LIN, outputs from different memory layers are resolved using tri-state logic.

The main benefit of D-LIN is reduction in number of TSVs. Since the number of TSVs in D-LIN is proportional to the number of masters $(p + d)$, and because *BankingFactor* is usually greater than one, the number of master channels will be less than the slave channels (m), hence, the reduction in the number of TSVs.

2.5 Dealing with 3D Integration Issues

This section presents architectural solutions to the issues related to 3D Integration of TCDMs using C-LIN and D-LIN.

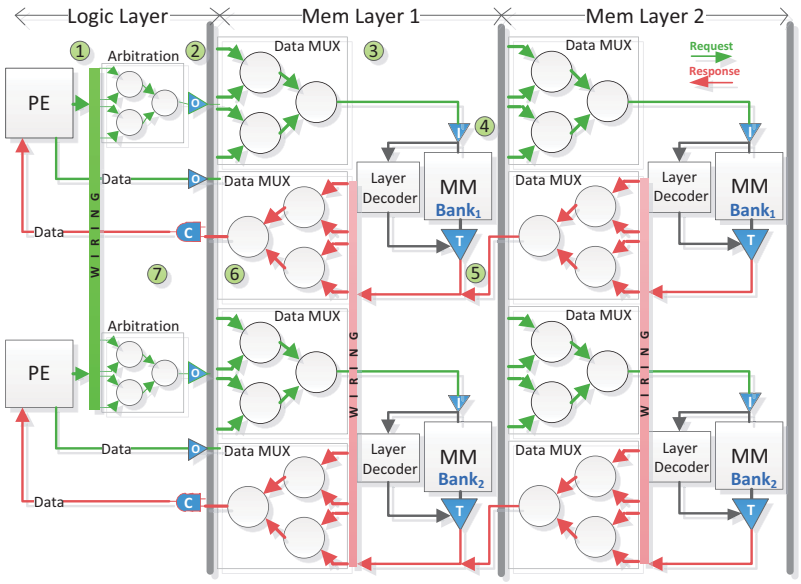


Figure 2.5: Block diagrams of the Distributed 3D Logarithmic Interconnect (D-LIN)

2.5.1 ESD Protection

The inter-die signal interfaces in a 3D-IC are vulnerable to electrical stress induced during stacking or testing steps. Approximately 60% of silicon IC failures are a result of electrical overstress or electrostatic discharge (EOS/ESD) [67]. To cope with these issues, IOs passing through the TSVs will be protected by input and output protection circuitry. As illustrate in Figure 2.4 and Figure 2.5, four types of IO Drivers are designed for this purpose: O and T are simple and tri-state output buffers respectively with reverse diodes for protection at their outputs, and I and C are input buffer and clamp circuits with protection diodes at their input stages. The O and C cells require level-shifters as well, since they may operate between two different voltage domains. All these cells have been adopted from conventional IO Pad Drivers and are tuned to drive much less capacitance of stacked TSVs of at most 8 layers. To avoid a long chain of buffers, the I and T cells are placed out of the chain at the inputs and outputs of each memory module (See Figure 2.6.a). This way, the signals do not need to travel through multiple buffers to reach a memory module.

2.5.2 Boot-time Configuration

In order for the memory layers to have identical layouts, boot-time configuration circuits are required to assign unique *LayerID* values to each memory die. For this

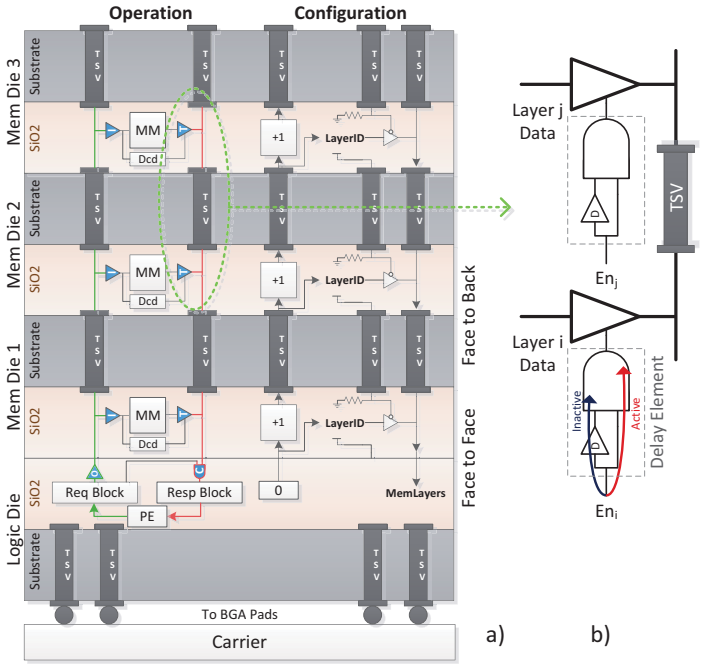


Figure 2.6: Structure of 3D stacking and boot-time configuration circuitry (a), delay elements to remove the high-current glitches (b).

purpose, assuming via-first technology [22], *LayerID* is incremented in each layer and sent to the next memory layer (See Figure 2.6.a). And in order to provide the total stacked memory size to the operating system, the last layer is identified by means of pull-down resistors, and its *LayerID* is returned to the logic layer as the number of memory layers.

2.5.3 Process/Voltage/Temperature Variations

The importance of process, voltage, and temperature variations intensifies in 3D circuits, since the dies from different process corners may be stacked over each other, and timing critical circuits such as the clock distribution network have to operate correctly under these conditions. One problem with such issues is the appearance of high current glitches on the outputs of tri-state drivers. As Figure 2.6.b illustrates, only one of the drivers should be active at a time, however, because of variations, one layer may start driving the bus before another has stopped, therefore, high current glitches will appear on the output bus which may degrade the chip's life and performance. In order to solve this issue, the driver in each layer should guarantee that it will return to inactive state before any other starts to drive. The simple delay elements illustrated in Figure 2.6.b can serve for this purpose, and by adjusting the delay between activation

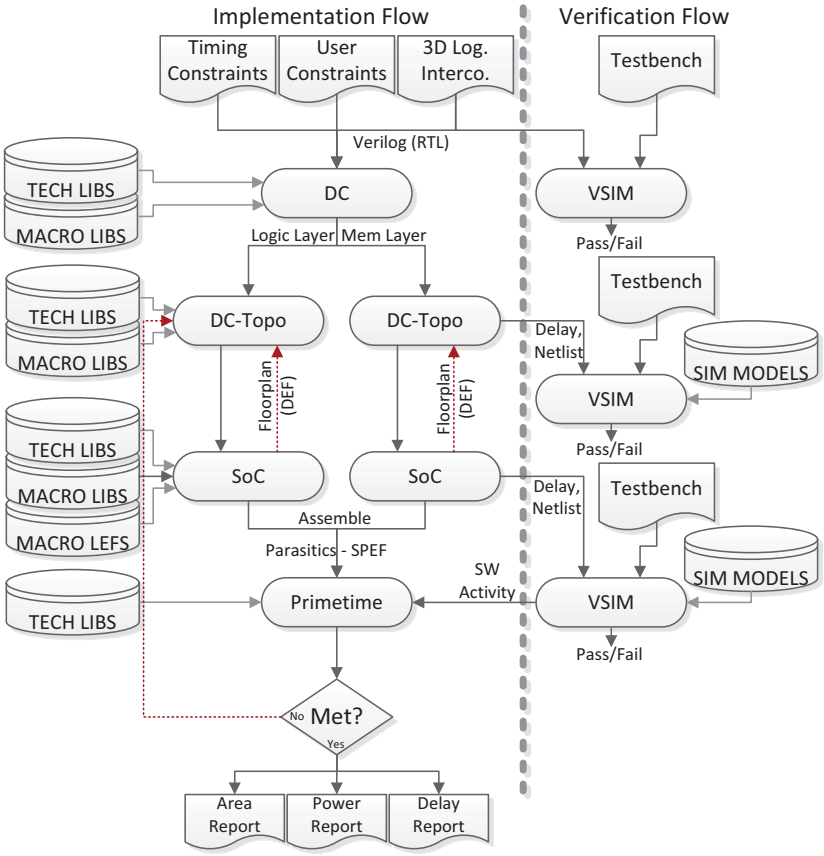


Figure 2.7: Overview of the hierarchical flow for design of C-LIN and D-LIN

and deactivation the glitches can be completely eliminated.

Another issue is the clock skew among memory layers which has been analysed thoroughly in [68], [69]. Moreover, synchronizers can be utilized between layers to correctly transfer data between clock domains [70]. Clock trees with tunable delays and phase detectors can also automatically remove the clock skews [71]. In order to maintain the combinational nature of the interconnection network, we utilize only the simplest method which is increasing clock margins during the Clock Tree Synthesis (CTS) phase.

2.6 Experimental Results

In this section, we discuss the experimental results for the low-latency networks in terms of timing performance and silicon area. Our baseline 2D-LIN platform is a multi-core

system composed by 16 (STxP70[1]) PEs that share the on-chip TCDM with 32 memory banks (*BankingFactor*=2) each having a size ranging from 8 KB to 64 KB. While in C-LIN and D-LIN, bank size is fixed (8 KB), and memory size increases modularly by changing the number of stacked layers from 1 to 8. For physical implementation we followed the hierarchical design flow illustrated in Figure 2.7. After a preliminary synthesis and timing budgeting of the whole design; topographical synthesis and place and route (P&R) are performed separately for logic and one memory layer. Then, the layers are assembled together and a capacitive load of 30 fF is used to model each TSV [22] (The capacitance of the micro-bumps is negligible compared to the TSV itself (less than 2 fF) [72]). Finally for the sign-off step, post place&route net-list along with the parasitic are fed into Primetime and a multi-corner static timing analysis (STA) is performed. If the obtained results are not suitable, the flow should iterate once more with possible changes in constraints.

We explored different configurations in terms of memory size embedded in the design, with metrics derived from the state-of-the-art technology and tools. Our design flow is based on the STM CMOS-28nm Low Power technology library, with a Multi V_{TH} synthesis flow with Synopsys Design Compiler Graphical (2011.09), and Place and Route with Cadence Encounter Digital Implementation (10.1), and the sign-off tasks in Primetime (2011.09). We assumed that memory dies in the 3D designs are stacked on top of the logic die, which provides power supply, clock, and data/control signals to them. The logic die has been designed using 10 metal layers, while this number has been reduced to 7 in memory dies because of lower routing complexity[2]. Memories and PEs have been implemented using pre-designed hard macros. As can be seen in Figure 2.6.a, the first memory layer is stacked over the logic layer using the face-to-face technology, as the others use face-to-back stacking. This eliminates the need for TSVs between the first two layers. In addition, the operating voltage of the memory layers has been increased slightly over the logic layer. This allows for removal of the level-shifter inside the C cell, and a reduction in its layout size and delay (Considering the fact that for 32 memory banks about 1000 C cells are required). Lastly, we implemented our solutions with two different bonding techniques: Micro-bumps and Cu-Cu Direct Bonding [74].

In this section, first we try to present the benefits of the logarithmic interconnect

[1]STxP70 is a cost effective 32-bit ASIP RISC core implemented using a 7-stage pipeline for reaching 600 MHz, which can execute up to two instructions per clock cycle (dual issue) [44].

[2]This reduction would lead to a significantly reduced mask and production cost, which is an example of how 3D integration enables cost optimization [73].

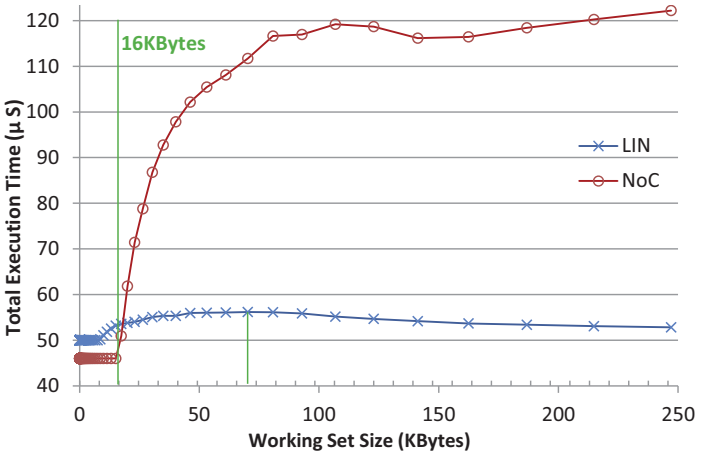


Figure 2.8: Performance comparison between NoC and LIN under different working set sizes.

in comparison with existing interconnect solutions, then we explore the design alternatives. Finally, we discuss the obtained results.

2.6.1 Comparison with Other Topologies

First, we compare LIN with an extremely high-performance NoC to show the superiority of low-latency interconnects in processor-to-L1-memory context. We modeled our baseline LIN(16×32) and a Mesh-NoC(4×4) presented in [75] in a home made cycle accurate trace simulator, fed with memory traces from MPARM [76] running a 16 PE configuration. For LIN, we assumed a clock frequency of 400MHz, Pseudo Round-Robin arbitration, and single cycle access to memory banks. While for the NoC we assumed a clock frequency of 5GHz (this is very optimistic for a low-power process such as STM CMOS-28nm) and a flit size of 32bits. We use pipelined 6-port switches [75] *(N, S, E, W, P, M)* with fall through latency of 5 clock cycles, and we assume the link latency to be 1 cycle. X-Y routing is assumed, as well. In our comparison we use memory banks of 8KB from the STM 28nm CMOS technology, with an access time of 1ns. We have attached two of these banks to each NoC switch aggregating a total of 32 banks. In the first experiment, each PE sends uniform traffic to the address range of $[HomeBank \pm WorkingSetSize/2]$, for 10000 transactions. As Figure 2.8 illustrates, when *WorkingSetSize* is small, every PE accesses its home bank only and NoC performs slightly better than LIN. However, as PEs start to access remote banks, NoC's execution time increases rapidly. While on the other hand, LIN's execution time recovers after an increase, because of load distribution among multiple banks.

Table 2.2: Performance comparison between LIN, NoC, and Bus executing different benchmarks.

Benchmarks		FAST	CT	SIFT
LIN	Execution Time (ms)	5.59	79.89	4464.07
	AMAT (ns)	6.54	6.47	6.46
NoC	Execution Time (ms)	8.21	106.92	4943.43
	AMAT (ns)	7.53	7.09	6.57
Bus	Execution Time (ms)	46.40	730.51	30799.99
	AMAT (ns)	81.30	82.40	81.90

For the second experiment, three embedded parallel image processing benchmarks [77] have been executed, and a time-division multiplexing bus has been modeled in our simulator, as well. Table 2.2 presents the results, where FAST is a corner detection algorithm, SIFT is an scale invariant feature transform, and CT is a color tracking algorithm. In all cases LIN performs better than NoC, and bus results are an upper bound to the execution time since all accesses result in contention. The reason for this advantage is that even though mechanisms such as speculative and look-ahead routing allow for a bandwidth of about 1 $flit/cycle$ in NoCs, yet they are not beneficial in processor-to-L1-memory context. Because traffics do not have a bursty nature, and small packets can not benefit from the huge bandwidth provided by NoC switches. In addition, memory access latency is critical in this context since it can lead to stalling the pipeline of the processors. Also it should be noted that there is an opportunity to further optimize the speed of LIN using custom circuit techniques such as [62]. However, this will result in incompatibility with standard technology libraries provided by silicon foundries and remove the configurability features of LIN.

Next, table 2.3 compares the post-layout area between LIN (16×32) and three other NoCs. All results have been scaled to 28 nm technology, and NoC switches have been scaled to flit size of $F = 32b$ by a factor of $(F_2/F_1)^{1.8}$ (extracted from Orion 2.0 [78]). As can be seen, NoCs require much larger areas than LIN, which is due to the large number of memory elements used in them. This is while LIN is purely combinational with no pipelining or buffering elements.

2.6.2 Design Alternatives

This subsection presents the results of timing performance and silicon area for our three designs: 2D-LIN, C-LIN, and D-LIN; implemented using two different bonding

Table 2.3: Comparison of post-layout area between LIN and NoCs.

Interconnect	Cardinality	Area (mm^2)
LIN	(p=16,m=32)	0.09
NoC-3.6GHz [79]	4x4	0.29
MIRA (3DM) [80]	4x4	0.40
MIRA (3DM-E) [80]	4x4	0.98
NoC-5.1GHz [75]	4x4	1.02

techniques. The area for PE hard macros (STxP70) is about $0.25mm^2$ and for each 8KB memory bank about $0.02mm^2$. Also for the Micro-bumps a minimum pitch of $40\mu m \times 50\mu m$, and for the direct bonding a more dense pitch of $10\mu m \times 10\mu m$ have been used [74]. In addition, as a corner case, the ESD protection circuitry has been removed from the direct bonding technique. The resulting layouts after full placement and routing for 2D-LIN and D-LIN (Cu-Cu direct bonding) are depicted in Figure 2.9.

Figure 2.10.a illustrates the silicon area (mm^2) for 2D and 3D implementations. As can be seen, the 2D die size increases with the embedded on-chip SRAM, except for the first two configuration, where PE obstructions and design geometry dominate the total design area (See also Figure 2.9.a). As the number of memory banks increases, total area grows, and large channels should be allocated for wires to reduce the routing congestion. Relative distances increase, and a massive number of buffers are inserted by the synthesis tool. In the 3D configurations, memory dies are equipped with a large number of TSVs, therefore a large portion of the die is allocated for TSV placement (routing obstruction and keep-out region for placement). This is illustrated in Figure 2.9.d. It should be noted that in this experiment, C-LIN contains 2688 TSVs, while, in D-LIN this number is reduced by 47% to 1424. The effect of TSVs is intensified when Micro-Bumps are used because of the large pitch.

Timing results are depicted in Figure 2.10.b where 8 memory layers have been stacked for 3D configurations. It can be seen that, C-LIN and D-LIN improve the performance over 2D-LIN with the same memory size by small factors of 6.7% and 3.7% respectively, while it is usually believed that TSVs can greatly reduce wire length and improve clock frequency. In order to explain this, consider the 2D planar design with 2MB of memory, for which we obtained a maximum frequency of 324MHz. While, for the C-LIN design with Cu-Cu direct bonding, we obtained a maximum frequency of 348MHz. If we assume that TSVs are ideal with zero capacitance, we can obtain a maximum frequency of 433MHz for the 3D design (which is comparable with the

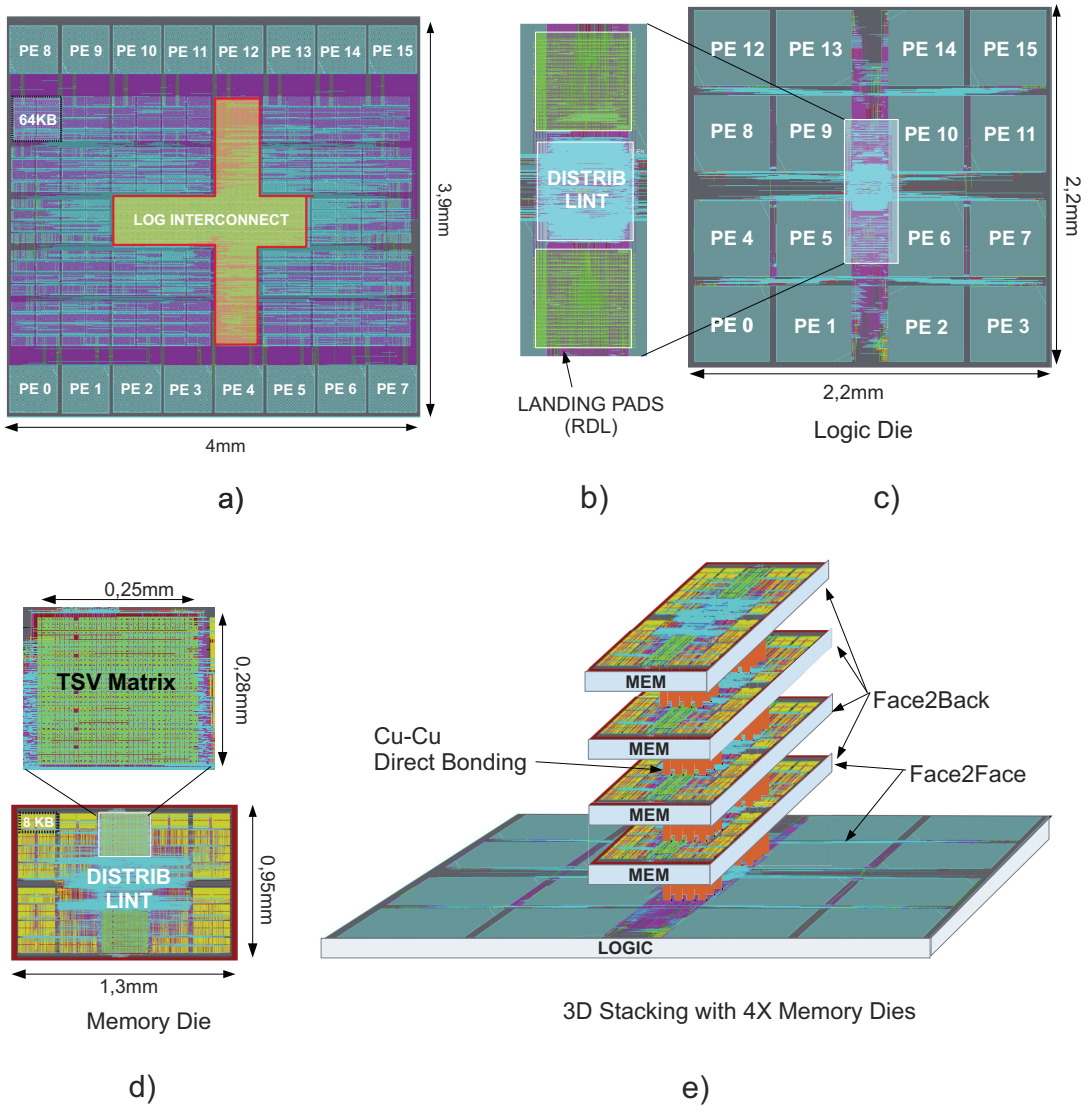


Figure 2.9: Physical implementation of the designs: 2-LIN with 2MB SRAM (a), Details of the landing pads on redistribution layer (RDL) in D-LIN (b), Logic Die of D-LIN with Cu-Cu Direct bonding (c), Memory die of D-LIN with details of the TSV Matrix (d), 3D Stacking with 4 stacked memory dies (e).

frequency of a 2D design with 256KB of memory). Comparing this clock frequency with 348MHz for C-LIN gives that 0.56ns of the critical path is devoted only to driving the TSVs, or in other words, TSVs and their drivers drop the performance over the ideal case by a factor of 24% (See Figure 2.11). This situation can be further explained considering the 30fF capacitive load of TSVs, resulting in a total load of 240fF in a stack of 8 TSVs, which is roughly equal to 4mm of $Metal8$ wire having a coupling capacitance

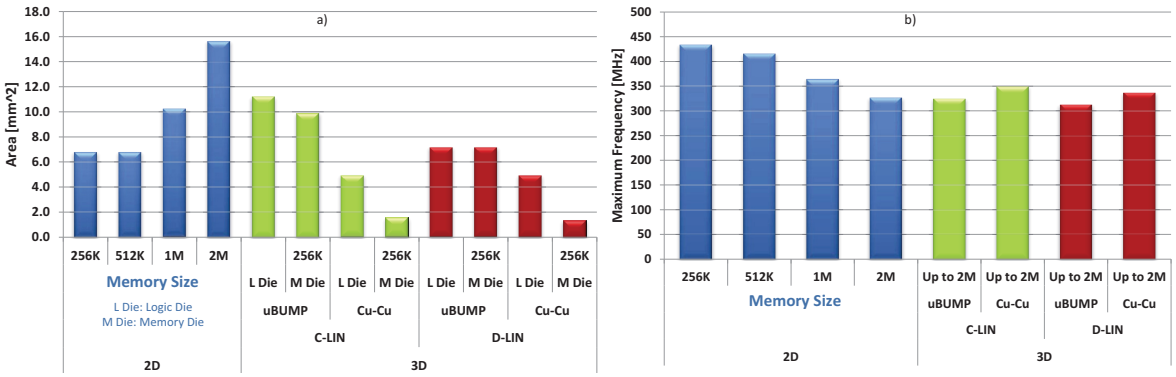


Figure 2.10: Comparison of silicon area (mm^2) (a), Maximum achievable frequency (MHz) (b).

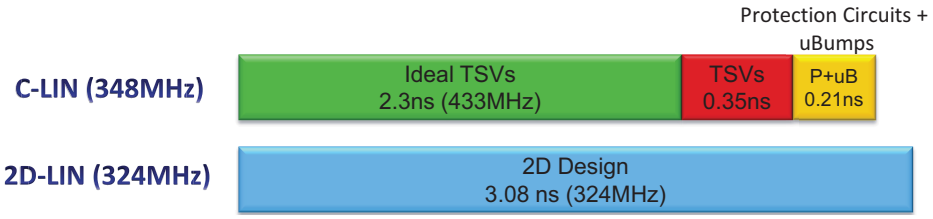


Figure 2.11: Effect of TSVs and their drivers on the critical path of the 3D designs.

of $66fF/mm$. This explains that current TSVs are not yet scaled enough to provide a major performance boost over 2D planar designs. Moreover, from comparison of results between micro-bumps and direct copper bonding without protection circuits, it can be estimated that micro-bumps and protection circuits further drop the performance by a factor of about 9%, consuming 0.21ns of the critical path.

One last point to mention is that, System Latency in [63] is calculated as (Network Latency + Memory Access Time), and by maintaining a fixed *BankingFactor* (reduction in number of banks per layers by addition of new layers), it has been shown that both system and network latency decrease significantly. While, in our experiments we showed that a major contributor to the performance drop is the latency of the TSVs and their drivers, and in order to support our argument we performed timing characterization of the whole 3D stack considering the TSV loads and their driver circuits, and derived maximum achievable frequency directly from the post-layout results of the physical synthesis tool.

2.6.3 Discussion

As our results demonstrated, the area and speed of the TSVs including their protection circuitry in current 3D technologies are not yet much better that the on-chip wires. However, 3D TSV technology can help reduce the overall cost of the die stack significantly, by implementation of the memory dies at lower costs: Using reduced number of masks or different technology options (e.g. different thresholds or different oxide thickness for the memory transistors to minimize leakage) to have better memories compared to the ones that could be implemented on the same die as the logic [73]. Furthermore, long critical paths in our single cycle design may suggest that current TSVs can be more beneficial in higher levels of the memory hierarchy where latency is not critical and pipelining can break the critical paths (e.g. [55]). Lastly, it should be noted that, as the network gets larger, the P&R effects such as long wiring buffers and routing congestion become increasingly important, and we believe that delays will increase even more in larger designs. Also, it should be noted that in a real design back-end, multi-corner and possibly multi-mode analysis should be performed which will make convergence even more difficult. Therefore, we suggest to build processing clusters with tightly coupled memories using proposed LIN alternatives, and use NoC as another level of hierarchy for inter-cluster communication, to benefit from both low latency of LIN and scalability of NoC.

2.7 Summary

In this chapter we presented two synthesizable network architectures, C-LIN and D-LIN derived from the Logarithmic Interconnect (LIN), which can be integrated with 3D Stacking technology to provide access to tightly coupled shared memory banks stacked over multi-core clusters. Architectural simulation results demonstrated that in processor-to-L1-memory context, LIN outperforms both traditional NoCs and simple time-division multiplexing buses. We devised a modular design strategy which allows users to stack multiple memory dies and create different height stacks with identical dies, without the need for different masks for dies at different levels in the stack. The designs have been explored in terms of area and latency, and full layout results show that for large 2D designs the main problems are routing congestion, signal integrity, and the mask cost. Therefore, our proposed 3D designs offer better scalability, however, in terms of delay they are not so competitive with their 2D planar counterpart. This is mainly due to the fact that the pipelines of the processors are extremely sensitive

to the access latency of L1 memories, and current TSV technologies are not yet so competitive with on-chip wires. For this reason, small sized L1 memories are not beneficial in terms of performance to be moved towards the third dimension. This motivates us study the effectiveness of 3D integration in the L2 memories shared by multiple clusters in chapter 3.

Chapter 3

3D Stacking of L2 Scratchpad Memories

In chapter 2, we observed that 3D-integration in the processor-to-L1-memory context, with current technologies, can only provide higher flexibility, modularity, and possible cost reduction opportunities. But from performance point of view, it can be even harmful due to relatively small sizes of the L1 memories and high sensitivity of the processor's pipeline to their memory access latency. In this chapter, we focus on out of the cluster L2 scratchpad memories, instead. We believe that they are more suitable candidates for 3D integration because of their large required size and higher tolerance to latency and its variations. We present a synthesizable 3D-stackable L2 memory IP component (called 3D-NUMA), which can be attached to a cluster-based multi-core platform through its Network-on-chip Interfaces (NIs), and provide high-bandwidth memory access with low average latency. 3D-NUMA is a scalable non-uniform memory access (NUMA) architecture which allows stacking of multiple identical memory dies, supports multiple outstanding transactions, and achieves high clock frequencies due to its highly pipelined nature. We implemented our design with STMicroelectronics CMOS-28nm Low Power Technology, and several experiments are performed to evaluate it from different aspects.

3.1 Motivations and Challenges

3D Integration has been explored in academia and industry for over a decade now, and a wide variety of technologies, materials, and processes have been used for research and demonstrations. Several vertical interconnect technologies have been explored,

including wire bonding, microbump, contactless (capacitive or inductive), and through-silicon-via (TSV) vertical interconnect [20]. Among them, the TSV approach has gained popularity, due to the high interconnection density.

Complex System in Package (SiP) solutions offered by companies such as TESSERA [38], STATSChipPAC [40], Amkor Technology [36], and INVENSAS [37], address a potentially large need in the market and are being recognized as the next industry thrust. Heterogeneous integration, system miniaturization and flexibility, and block level testability are some of the several features offered by SiP solutions. In addition, they provide a path to integration of planar IC with 3D-IC technology [41]. TSV Silicon Interposer (TSI) is a good example of how heterogeneous dies with mixed technologies can be integrated at higher levels and greatly reduce die complexity and cost [39]. This type of "technology disaggregation" offers the opportunity to optimize the silicon technology for each individual IP, increases the capabilities of the products offered to the users, provides more memory options and storage per ASIC, and more importantly, reduces yield loss due to large dies [40]. On the other hand, with the miniaturization of packages and integration of different dies, some issues such as heat dissipation and power delivery have raised a lot of concerns. As stated in [81], 3D stacking has a higher impact on IR-drop than original 2D designs, since it inherently increases the resistance of the power delivery network (PDN) which directly impacts IR-drop. Moreover, stacking of dies over each other results in much higher power densities, and exacerbates heat dissipation issues [82].

In this chapter, we focus on L2 scratchpad memories for three-dimensional integration, because of their relatively lower sensitivity to access latency and its variations. Also because, most application processors and almost all mobile SoCs feature a large on-chip L2 memory which is shared by multiple cores. Snapdragon 800 Processors by Qualcomm with 2MB of L2 Cache [32], Exynos 5 by Samsung with 1MB of L2 Cache [33], and Keystone II by Texas Instruments with 4MB of Shared L2 memory plus 4MB of private L2 space configurable as cache or SPM [83], are great examples in this context. We present 3D-NUMA, an L2 memory IP designed for integration as a 3D stacked module, which can be attached to a cluster based multi-core platform through its NoC Interfaces (NIs) (See Figure 3.1), offering high-bandwidth memory access with low average latency. Our proposed IP is a synthesizable and scalable non-uniform memory access (NUMA) architecture which allows modular stacking of multiple memory dies with identical layouts using a single mask set, supports multiple in-flight transactions, and achieves high clock frequency, thanks to its highly pipelined nature.

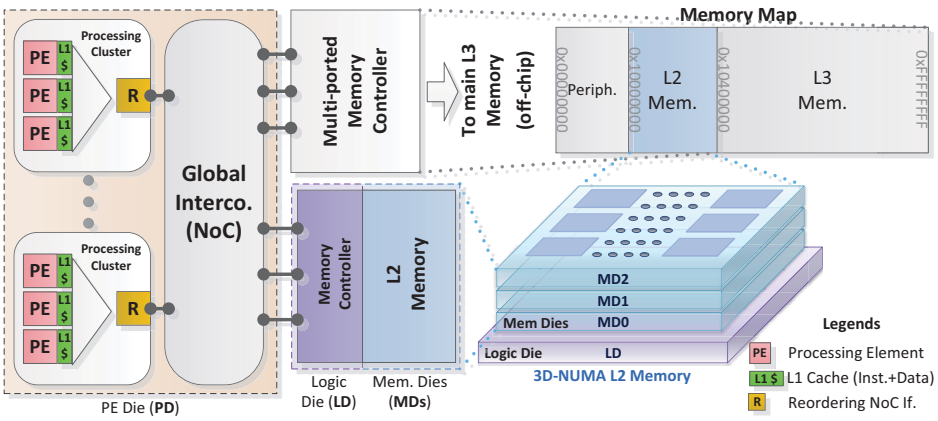


Figure 3.1: Overview of the target architecture and memory map for 3D-NUMA memory IP

This chapter is organized as follows: Related research efforts are discussed in section 3.2. An overview of 3D-NUMA is presented in section 3.3. Post-layout synthesis results are presented in section 3.5. Performance, power, and temperature analysis are performed in section 3.6 and section 3.7. Issues related to packaging and manufacturing are discussed in section 3.8 and section 3.9, and lastly, a summary of the obtained results is presented in section 3.10.

3.2 Related Works

Advanced packaging technologies provide new opportunities for heterogeneous integration, power delivery, cost optimization, and thermal management. Stacked Chip Scale Packaging (SCSP) of Amkor Technology [36] is one such example which provides several different 2.5D/3D options for integration of heterogeneous dies in a package. Among other packaging technologies, Dual DRAM Package (DDP), Dual Face Down (DFD), and Quad Face Down (QFD) [37][38] with the main target of DRAMs provide complex forms of wire-bonding which may be adopted even for other levels in the memory hierarchy. Technologies such as TSV Silicon Interposer (TSI) [39][40] and wafer reconstitution [41] provide even more flexibility in hybrid 2.5D/3D stacking. TSIs enable stacking of different dies on both sides to achieve a better utilization of space and to facilitate heat transfer of high power chips. Wafer reconstitution provides electrical connections from the chip pads to the interconnects by means of an artificial wafer. Redistributed Chip Packaging (RCP) [41] developed by Freescale Semiconductor offers scalable chip-scale packaging and multi-die heterogeneous integration. In addition,

Package on Package (PoP) stacking is supported in RCP by means of Through-Package Vias.

3D stacking of DRAM memories as the last level in the memory hierarchy is orthogonal and complementary to the focus of this chapter, and will be studied in chapter 4. However, 3D stacking of SRAM memories can provide more flexibility, opportunities for process optimization, and simplified supply chain. Since dies are homogeneous from a technology viewpoint, i.e. they can be manufactured in the same fab as logic dies. Needless to say, given their low density and high cost, SRAM based memories are obviously not a viable DRAM replacement for main memory, and they should be used in lower levels of the memory hierarchy. We should add here that, embedding DRAM memories in the lower levels of memory hierarchy is also another design alternative. Tri-gate CMOS Embedded DRAM (eDRAM) designed in 22nm technology by Intel [18], and the 45nm SOI eDRAM by IBM [84] are two examples which can offer better area utilization, performance, and even power consumption compared to the SRAM cells in the same technology nodes [85][86]. However, these technologies require special process options and they are expensive compared to the state-of-the-art memories. In our design we use industrialized SRAMs, nevertheless, our proposed architecture for L2 memory can be easily adapted to use embedded DRAM, as well.

3D stacked caches with wide I/O interfaces [28][29], and 3D stacked non-uniform cache architectures (NUCA) [30][25][31] are alternative solutions for moving towards the third dimension. In [28], the authors have demonstrated that implementing the memory bus between a L2 cache and an on-chip main memory as wide as a cache line, which operates at core's clock frequency, can provide the maximum bandwidth that the L2 cache can consume and, thus, contribute to a large gain in system performance. In [29] a 3D stacked SRAM cache with wide vertical I/O interconnections has been fabricated at $0.18\mu m$. However, in spite of all advantages in 3D stacked caches with wide I/O interfaces, a centralized shared memory still lacks scalability [87]. On the other hand, NoC based 3D stacked NUCA brings a scalable and modular communication infrastructure. In 3D stacked NUCA, the stacked cache is divided into multiple banks with different access latencies according to their locations to cores [30]. Connecting each processing core to the 3D stacked NUCA cache separately through its own TSVs enables high bandwidth and parallel communications between cores and stacked cache banks. In [25], some advancements in trying to solve the DRAM issues are presented with real chip results for WIOMING Multiprocessor SoC. An innovative distributed caching mechanism is proposed to reduce memory access latency and external memory

bandwidth requirements. Lastly, [31] uses SRAM row cache to improve both performance and energy in a 3D stacked DRAM. While 3D stacking of caches is also an alternative solution, in this work we focus only on design of large on-chip memories which are mapped in the address space to accommodate data structures which must remain on-chip for sake of energy efficiency, and that are not well managed through caches.

3D stacked scratchpad memories (SPM) have been studied in [64][65][55]. In [64], the authors proposed a configurable memory die that consists of many uniform memory elements, which are connected to each other with a switch-based 3D mesh interconnection network. In [88], customizable redistribution layer (RDL) routing for a configurable 3D stacked memory has been proposed. The RDL enables connecting each core and memory cell without any switch connection. In [65] a prototype of 3D stacked SPM has been published. It is a two-layer 3D IC, where the logic die consists of 64 general-purpose processor cores running at 277 MHz, and the memory die contains 256 KB SRAM. Each processor core is directly connected to each 4 KB of SRAM scratchpad data memory. All these works focus on 3D stacking of private memory banks. While, in [55], a logarithmic interconnect provides low latency access to a stack of shared L2 SPM dies. The main difference between this work and the presented works is that we propose a high performance and scalable 3D non-uniform memory access (NUMA) architecture for L2 SPMs, which allows modular stacking of multiple memory dies with identical dies with a single mask set. Moreover, we have implemented our proposed solution using the state-of-the-art technology libraries and performed several experiments on power, performance, temperature, etc.

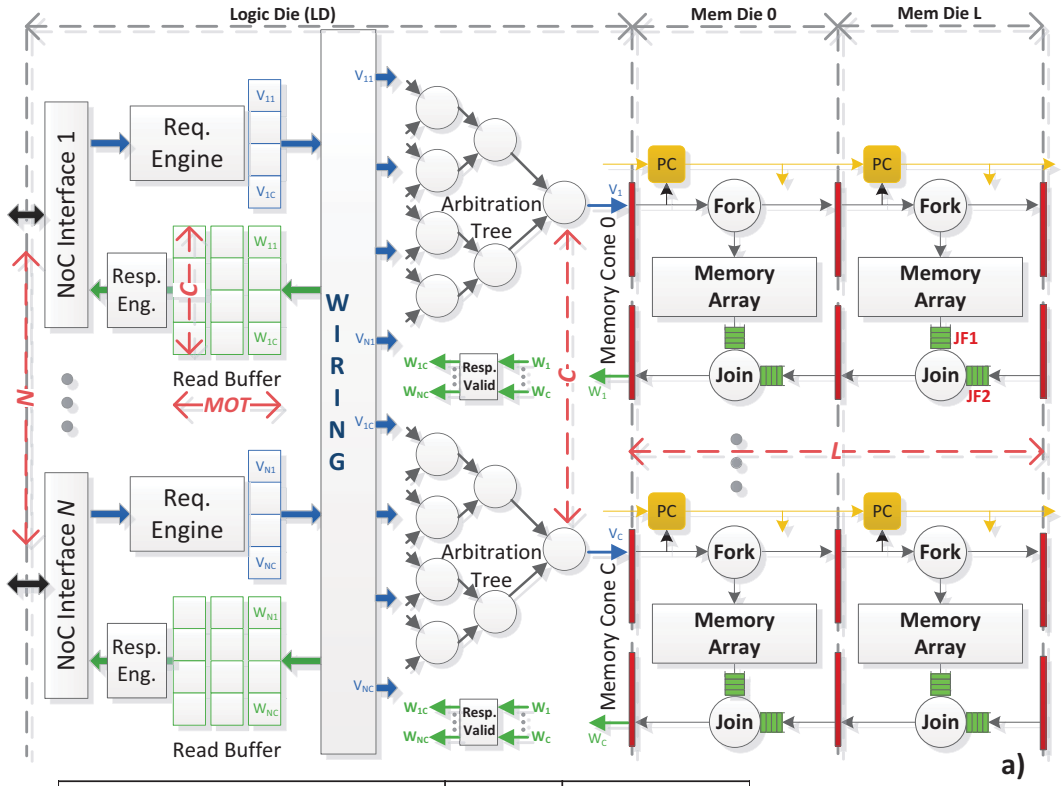
Focusing on die stack ordering strategies, several different configurations are possible to be used for a 3D stack consisting of memories, logic, and processing elements (PEs). 3D stacked memories initially had their memory dies placed over the PE die. For example, in [64] one layer of scratchpad memory is stacked over the PE die. In [89] multiple levels of DRAM dies are stacked over a PE die composed by processors and caches. In [90] DRAM dies are stacked over an interposer, while the logic die is placed under it. And even in the initial proposal of Hybrid Memory Cube (HMC) by micron multiple DRAM dies have been stacked over a logic die [91]. One drawback with these solutions is the difficulty in heat dissipation of the PE or the logic dies. Most recently, 3D configurations with PE dies stacked over the memory dies have been proposed. In [71] two face-to-face core and cache dies are stacked back-to-back and then placed over multiple DRAM dies. In [82] similarly, PE die is stacked over a cache die and then

placed on top of multiple memory dies. A first-order thermal analysis shows improved heat transfer in this configuration. In [92], an unthinned PE die is stacked over multiple thinned DRAM dies, with a heat-sink placed over them. For power delivery to the PE die, Through-DRAM TSVs are utilized. Despite heat improvement in all these solutions, they aggravate the very important problem of power delivery to the high-power dies (i.e. PE and logic dies) stacked on the top levels. Power TSVs consume a large percentage of die area, and more importantly, IR-drop and voltage droops in the power distribution networks increase to a great extent [81].

One promising solution which can solve the drawbacks of the two mentioned configurations, is through introduction of a double sided silicon interposer to the stack [93] (see Figure 3.18). This way, high power dies can be stacked on its top, while low power dies such as memories are placed on bottom. RDL metal layers on both sides of the interposer along with TSVs passing through it provide connections between the two sides. In [94] one such solution is proposed with a logic controller stacked on top of a substrate. While the whole DRAM stack along with its controller are placed on the bottom of the substrate. Similarly in [95] packaging requirements for such DRAM stack design is investigated, and even a back-side heat slug is utilized for heat dissipation of the low-power dies on the bottom side. To our knowledge, our work is the first one with physical implementation of a 3D stacked L2 scratchpad memory and an extensive evaluation of several different parameters.

3.3 3D-NUMA Memory IP

3D-NUMA is a 3D L2 scratchpad memory designed to be attached to cluster based multi-core platforms with a global NoC connecting all the clusters, and each cluster composed by multiple tightly coupled processors (illustrated in Figure 3.1). This memory IP is well-suited for serving L1 cache refill/write-back commands, since it has been designed to serve load and store packets of different sizes (up to 64 Bytes). It should be noted that 3D-NUMA is mapped in the main address space and it supports caching similar to the main memory, nevertheless, actual caching mechanisms (e.g. directory/snooping mechanisms) are not implemented in this work, and pre-recorded traces are used for performance analysis (See section 3.6). The word-level-interleaved organization utilized in this memory system allows for breaking a *Load64Bytes* command into *Load8Bytes* commands and dispatching them to 8 parallel memory cones (See Figure 3.2.a). This way, 3D-NUMA can offer much higher bandwidth than simple



Parameter	Symbol	Values
Number of NoC Interfaces	*N*	1, 2, 4, ..., 16
Number of Memory Cones	*C*	1, 2, 4, ..., 16
Maximum Outstanding Transactions	*MOT*	1, 2, 4, ..., 16
Number of Stacked Memory Dies (Layers)	*L*	1, 2, 4, 8
Size of Each Memory Array	*S*	Any (32 KB)
Width of Data Bus	*W*	Any (64 bits)
Width of Address Bus	*A*	Any (32 bits)

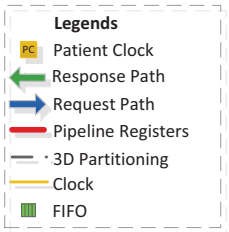


Figure 3.2: Block diagram of the 3D-NUMA memory IP (a) and its configurable parameters (b).

bank-level-interleaved memories directly attached to the NoC interfaces (See section 3.6 for detailed results).

An overview of the 3D-NUMA L2 memory IP is illustrated in Figure 3.2.a. When request packets of different sizes arrive at the NoC Interfaces (NIs) (*Load/Store*{*1, 2, 4,..., 64*}*Bytes*), the Request Engines (REs) break the input packet into smaller units called *chunks* and issues them in parallel to the Arbitration Trees (ATs), where a pseudo round-robin arbitration is performed among the requests arriving from different NIs (V_{ij} and W_{ij} in Figure 3.2.a show the wirings for requests and responses, respectively). Winners enter the memory pipeline, while the losers wait for another cycle behind

the arbitration trees. Each request travels through the memory pipeline, and in each memory die a partial address check is performed in *Fork* to identify whether the request belongs to that particular memory die. If matched, memory access is performed and a response is returned in the response path through the *Join* modules. Response paths are shared among the Read Buffers, and simple return-address decoders issue valid signals (resp. valid component) to the destination. Since the response chunks arriving from different memory cones may arrive out of order and at different times, a data structure called Read Buffer (RB) is utilized to merge them, build response packets to original requests, and serialize them through the NI. It should be noted that the access time of the memory dies increases with their indices (Non-Uniform Memory Access [NUMA] behaviour), since the memory dies are separated by pipeline registers and packets flow through these registers in each cycle. This feature allows for scalability, facilitates stacking of new memory dies with a single mask set, and modularly increases the memory size without affecting the clock frequency (Effect of the number of stacked dies on memory access time is studied in section IV). One should note that, such change in a flat die would require a complete silicon re-spin.

The key property of this soft IP is configurability through several parameters illustrated in Figure 3.2.b. N is the number of independent NoC interfaces which are used to attach 3D-NUMA to a NoC (the design of which follows AXI bus standard [96]). C is the number of parallel memory cones. This parameter defines the maximum possible number of words which can be fetched in parallel during a load operation. MOT defines the maximum allowed in-flight transactions inside the memory system. This parameter directly affects the depth and complexity of the read buffers (described in the next subsection) while it has no effect on the memory pipeline and the other components. L is the number of the stacked dies. S defines the size of each memory array, and finally, W and A define the widths of the data bus and address bus, respectively.

One last point to mention is that this memory organization does not maintain the order of the input packets, and may reorder them due to its NUMA nature. Therefore, existence of reorder-buffers at the refill ports of L1 caches or the DMA engines in the clusters is mandatory. This is not an unrealistic requirement, since most advanced cache interfaces and DMAs handle Out Of Order (OOO) NoC responses [96].

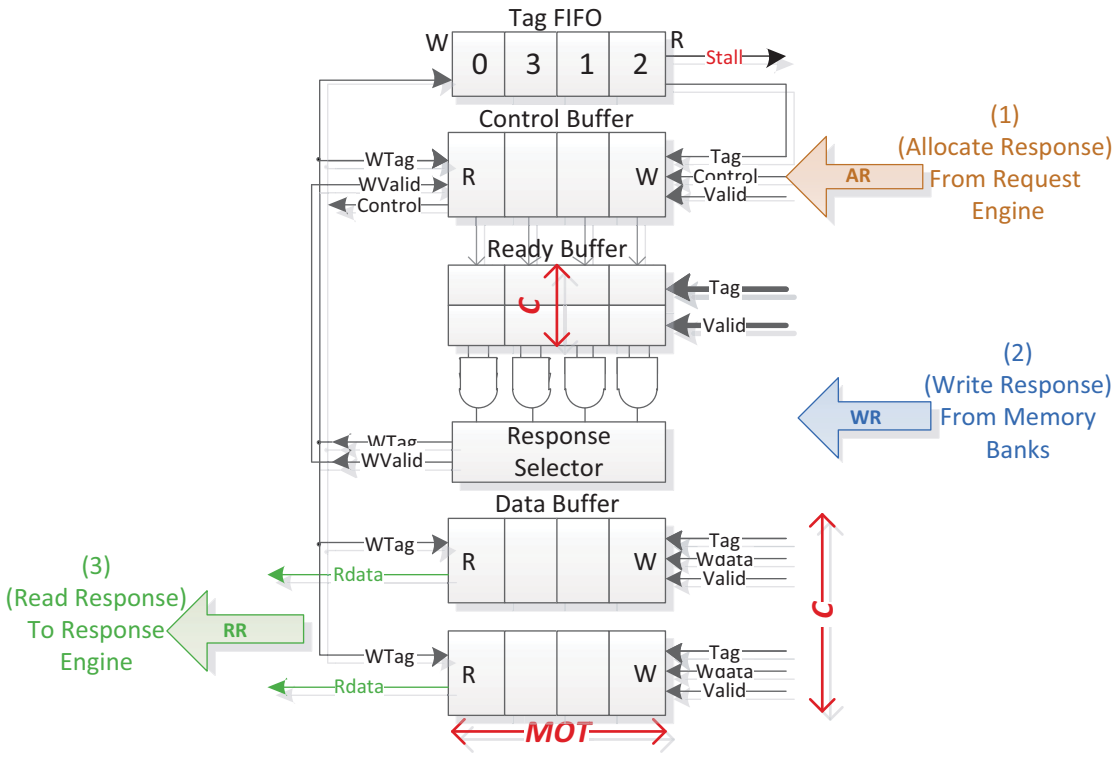


Figure 3.3: Implementation of the Read-Buffer for $C = 2$, $MOT = 4$.

3.4 Network Operation

When a request packet arrives at one of the *NoC Interfaces* illustrated in Figure 3.2.a, it is forwarded to the *Request Engine* through a small queue. The request engine tries to allocate a location for this packet inside the *Read Buffer*, and if succeeds, it will the grant the original request, otherwise stalls it. After successful allocation of the request in the *Read Buffer*, it will be broken into *chunks* (word-level-interleaved based on its lower address bits), and each chunk is sent to a separate memory cone and waits there for the arbitration tree in that cone to compete with requests from other NoC Interfaces. The winner chunk is allowed to enter the memory pipeline and move forward through the pipeline registers in each clock cycle. In the memory pipeline, *Fork* and *Join* components are responsible for flow control and routing of the request to the intended memory layer, and returning back the response to the NoC Interfaces. *Read Buffer* component is described in the next subsection and other components follow it afterwards.

3.4.1 Role of the Read Buffer

Read Buffer is one of the most important components in 3D-NUMA, as it serves for different purposes: First, it allows for supporting multiple outstanding transactions and decouples request and response paths completely, by accepting up to *MOT* requests and granting them while their responses are not ready yet. This helps utilize the bandwidth of the memory pipeline more efficiently. In addition, since response chunks from different memory cones may return at different times, RB stores them temporarily until they all become available. Finally, all the header and control bits of the input packet are stored in the RB to avoid propagating them through the whole memory pipeline. When response data returns from the memory pipeline, response packet is built using this stored information.

The schematic view of the *Read Buffer* is illustrated in Figure 3.3. When a request arrives at one of the *Request Engines*, it is allocated in the associated *Read Buffer* by assigning a new *Tag* to it from the *Tag FIFO* (If the Tag FIFO becomes empty, the request will be stalled). Next, all the header and control bits of the input packet are stored in the *Control Buffer* to avoid propagating them through the whole memory pipeline. Finally, the location associated with this Tag inside the *Ready Buffer* will be cleared. All these operations are performed in parallel in the step shown in Figure 3.3 as *Allocate Response (AR)*. A couple of cycles after issuing the request, the response chunks return from different memory cones. Read Buffer accommodates them based on their tag and partial address bits, and sets the ready bit for each one of them inside the *Ready Buffer*. This step is shown as *Write Response (WR)*. A response becomes ready when all of its chunks have arrived from the memory pipeline. Then, a First-comes-first-served (FCFS) arbitration is performed among all the ready responses inside the *Response Selector* module, and winner is sent to the *Response Engine* along with its header and control bits which are necessary to route back the response. Finally, its location is marked as empty for accommodation of new requests (*Read Response (RR)* step).

3.4.2 Flow Control Components

Flow control in the logic die of 3D-NUMA is based on request-grant handshaking and supports full bandwidth operation of 1 transaction per cycle, whereas, the memory pipeline has been designed in a grant-less and straightforward fashion. In fact, whenever a requests chunk enters the memory pipeline, a location for its response is already

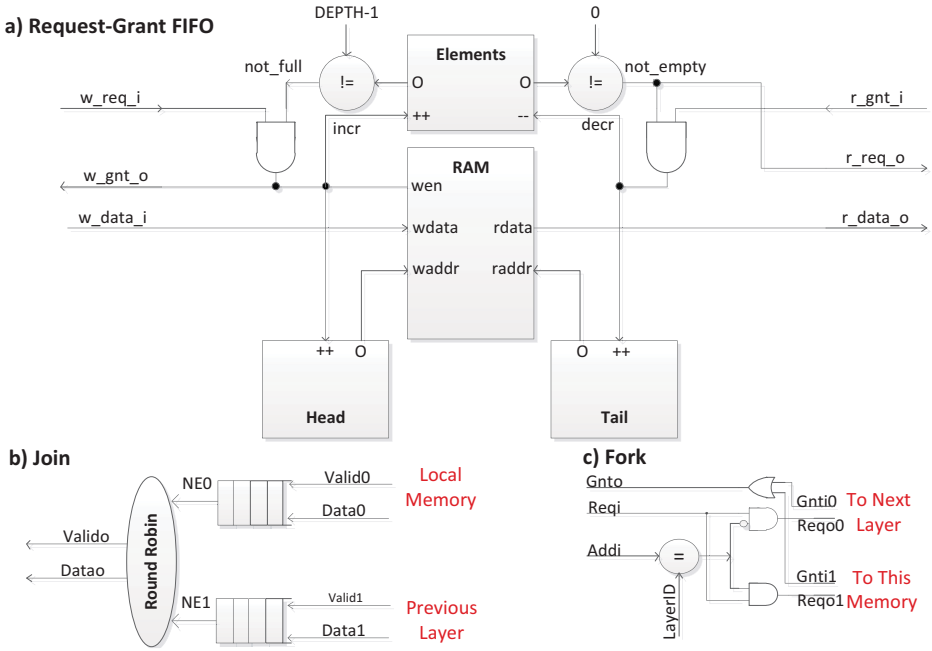


Figure 3.4: Schematic view of request-grant FIFOs (a), Join (b), Fork (c) components.

reserved in RB, therefore, there is no possibility that this request or its response get stalled. This is another benefit of RB which helps remove the grant signal from the memory pipeline, and greatly simplifies its hardware by replacing complex request-grant FIFOs with simple pipelining flip-flops (illustrated in red color in Figure 3.2.a).

Fork is a combinational module which simply compares the address of the request chunk with *LayerID* (index of current memory die/layer) and if matches, it sends the chunk to the memory cut on this layer, otherwise forwards it to the next layer. While, *Join* receives response chunks from this layer and the upper memory layer through small FIFOs designed for this purpose, and chooses between them in a round-robin fashion, and forwards the winner to lower layers. A schematic view of the Fork and Join components, as well as, the request-grant FIFOs is illustrated in Figure 3.4. *Request Engine* is responsible for decoding request packets and issuing them to the memory pipeline. *Load* packets from the NoC Interface fit in a single flit, since they do not have any payload. Therefore, Request Engine receives the whole packet in a single cycle and then issues it to parallel memory cones (Only aligned access is supported). While, *Store* packets have several flits, therefore, *Request Engine* issues requests to associated memory cones serially, as it receives each flit. Lastly, *Response Engine* receives response packets which are ready from *Read Buffer*, and serializes them into multiple flits to the

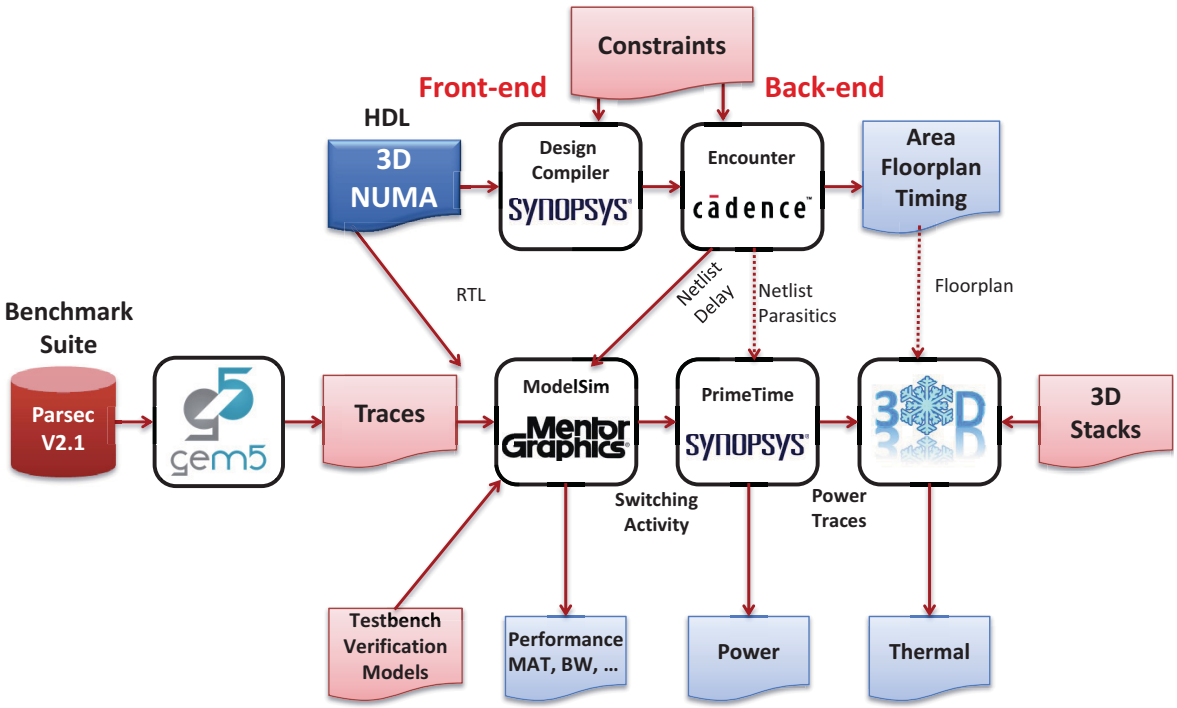


Figure 3.5: Experimental setup for design and exploration of 3D-NUMA

NoC Interface. In case of *Load* commands, a multi-flit packet is returned, while for *Store*, only a single flit acknowledge packet is returned. It should be noted that, three different arbiters are advocated in design of 3D-NUMA to resolve events happening in the same cycle. *Arbitration Tree* (See Figure 3.2.a) is a modified version of the Logarithmic Interconnect presented in the previous chapter which performs Pseudo-round-robin arbitration over the input requests, and is modified to support grant-based handshaking. *Join* module uses a simple Round-robin arbiter implemented as a toggle flip-flop. And a FIFO based implementation of First-come, first-served (FCFS) has been implemented inside the *Response Selector* (See Figure 3.3). All mentioned arbitration algorithms have been chosen carefully to be fair, otherwise they can result in starvation.

3.5 Design Implementation

An overview of the experimental setup for design and exploration of the 3D-NUMA memory IP is illustrated in Figure 3.5. The individual steps will be described in this section and the followings. Physical design of 3D-NUMA has been performed

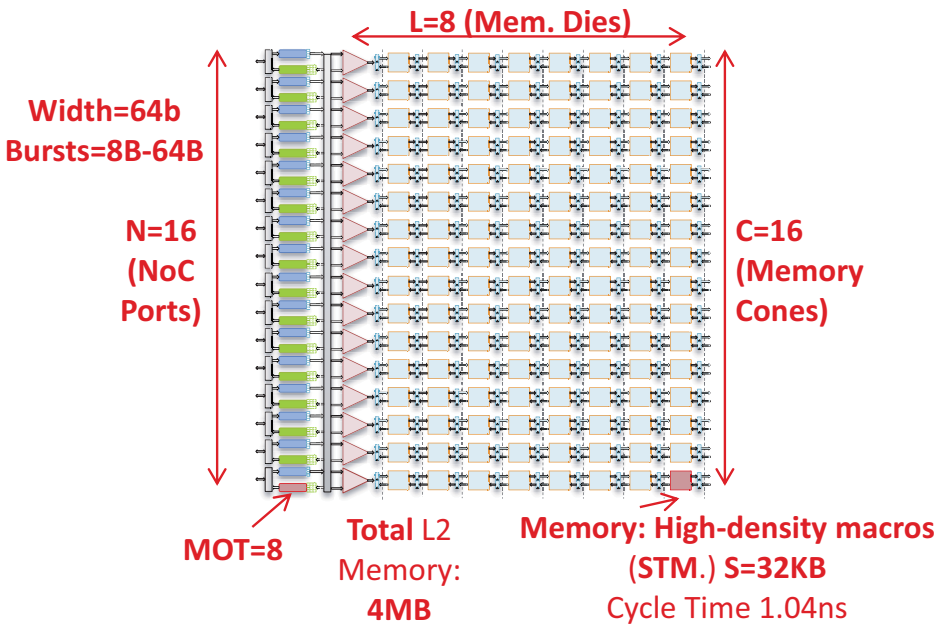


Figure 3.6: Schematic view of the 3D-NUMA design to implement

based on the STM Bulk CMOS-28nm Low Power technology library, with a Multi V_{TH} synthesis flow with Synopsys Design Compiler Graphical, and P&R in Cadence SoC Encounter Digital Implementation. The baseline 3D-NUMA memory IP consists of one Logic Die (LD) with 16 NoC Interfaces (N) and 8 (L) identical Memory Dies (MD) stacked over it. The number of parallel memory cones is 16 (C), maximum supported address width is 32 bits (A), width of the data bus is 64 bits (D), and the maximum number of outstanding transactions is equal to 8 (MOT) for each NoC interface. The schematic view of the design to be implemented is shown in Figure 3.6. For the memory arrays, high density industrial hard macros (S=32KB, W=64bits), provided by the STMicroelectronics company in the same technology, have been utilized. This is to provide an access granularity of 64-bits, while the burst-length can flexibly range from 8 Bytes to 64 Bytes. The access time of these memories is 0.786ns with a cycle time of 1.04ns. Up to a total of 4 MB of stacked L2 SPM can be provided by this IP. NoC Interfaces have been implemented using pre-designed hard macros, as well, and for the memory elements inside the Read Buffers small hard macros of 16 $\times$ 64 bits are utilized. All other components have been synthesized as Soft IPs. The logic die is designed using 10 metal layers, while this number is reduced to 8 in memory dies because of lower routing complexity. This configuration has been used for all experiments, unless otherwise stated.

3D clock distribution and delivery have been analysed thoroughly in [68][97], and

clock skew among layers can be dealt with synchronizers [70] or clock trees with tunable delays and phase detectors [71]; In this design we have assumed a synchronous 2.0ns clock with ample margins propagating through dedicated TSVs to all stacked memory dies. Clock skew has been handled through design margins similarly to the 2D case. For the TSVs we used via-first technology with Cu-Cu Direct Bonding technique and a pitch of $10\mu m \times 10\mu m$ [22] (as the state-of-the-art for high volume production-ready TSVs). A capacitive load of $30fF$ [22] has been used to model them, and an extra margin of 140 ps has been added to the output signals which are captured by flip-flops (FFs) in the adjacent memory dies (setup time of the FFs), during the timing analysis phase.

TSV fabrication yield is a crucial parameter in manufacturing yield and cost of the final stack. For this reason, use of failure detection and repair mechanisms is highly beneficial when the TSV process technology is not mature enough. In this design, we have utilized a low overhead TSV repair mechanism capable of providing up to 95% recovery rate with an overhead of 1 TSV per each block of 25 [98]. The effect of these TSVs on manufacturing yield and cost is studied in section 3.9. Moreover, area overhead of the redundant TSVs along with their detection and recovery circuits are considered in all other experiments. Due to the highly pipelined nature and enough margins on the 3D clock networks, 3D-NUMA can easily tolerate the extra latency introduced by the TSV repair mechanism.

Figure 3.7 illustrates a snapshot of the post place&route layouts. As can be seen, an area of 0.31 mm^2 is devoted to the TSV matrix (between LD and MD0) composed by 3088 TSVs with a pitch of $10\mu m \times 10\mu m$, and for connection of LD to PE die (PD) 16 groups of 228 TSVs with areas of 0.02 mm^2 are utilized. Placement and routing under the TSVs have been avoided to prevent thermal stress during fabrication [99]. We should remind that the dies have been designed as generic as possible to be used with different stacking configurations, therefore, in some of the configurations TSVs can be eliminated because of face-to-face stacking (e.g. TSVs between LD and PD in the MPI configuration in Figure 3.17.a). Each 32 KB memory array consumes about 0.062 mm^2 of silicon, and the logic elements and routing channels on the memory die add an extra overhead of 9% (apart from the area occupied by TSVs and power rings around the memory hard macros). We would like to mention here that our 3D design is capable of reducing die footprint by over 75% compared to its 2D counterpart. This can provide the opportunity to improve power delivery, IR-Drops, and manufacturing yield and cost. Of course this can happen only with enough maturity of the TSV technology

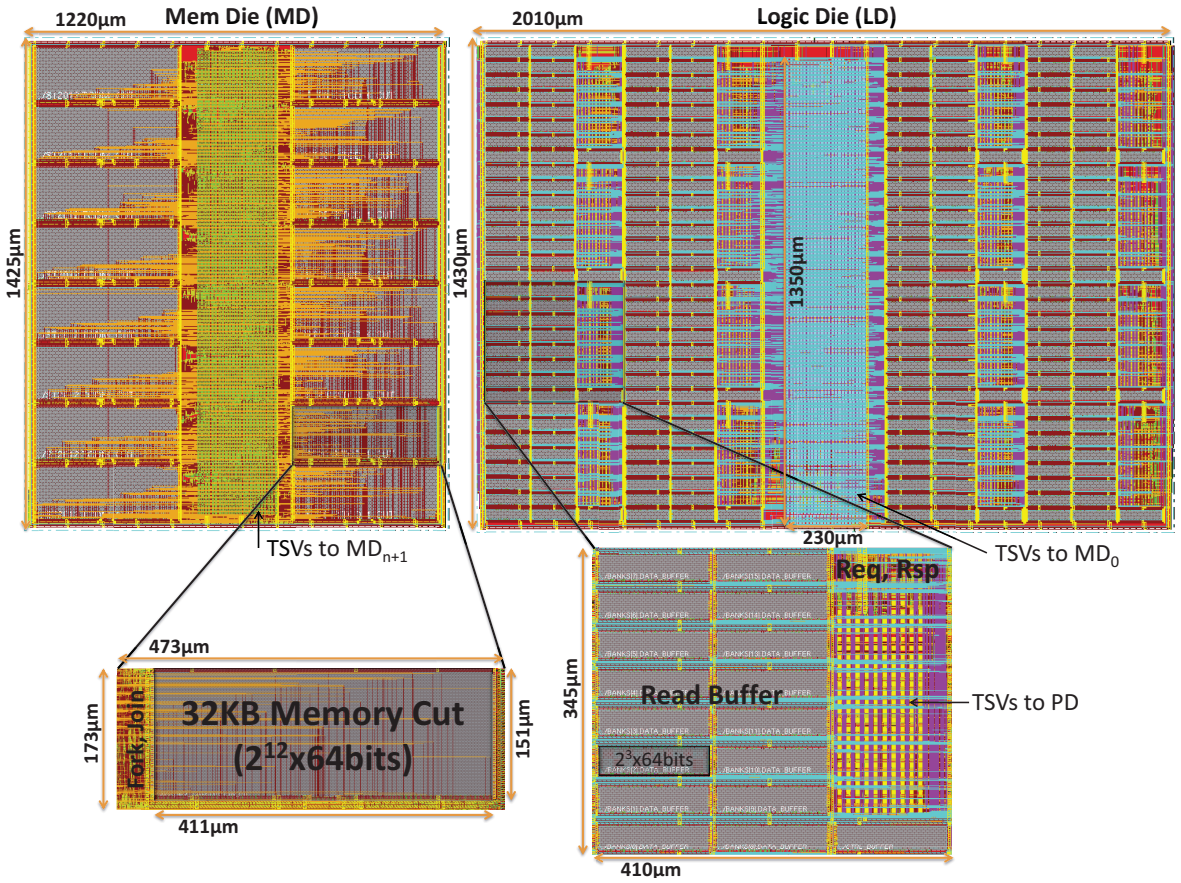


Figure 3.7: Physical implementation of 3D NUMA in STMicroelectronics CMOS-28nm Low Power Technology.

and the automated 3D CAD Tools. Lastly, post P&R timing results demonstrate that our baseline design can operate at 500 MHz in the slow process corner (SS) with an operating voltage of 0.9V, and a temperature of 0° C. This is limited by the access time of the memory arrays, while the logic components can operate up to 1 GHz.

3.6 Performance Evaluation

This section presents the performance evaluation and design space exploration results of the 3D-NUMA memory IP under different configurations and loads. Detailed cycle-accurate simulation has been performed in Mentor Graphics' ModelSim. *gem5* [100] simulation environment has been used to record memory access traces, which are then fed to ModelSim for trace-based simulation. *gem5* runs a full-system simulation of Alpha CPUs with Linux 2.6.27 kernel executing PARSEC V2.1 benchmark suite [101] on medium sized inputs, and a fixed number of traces are gathered only for the parallel

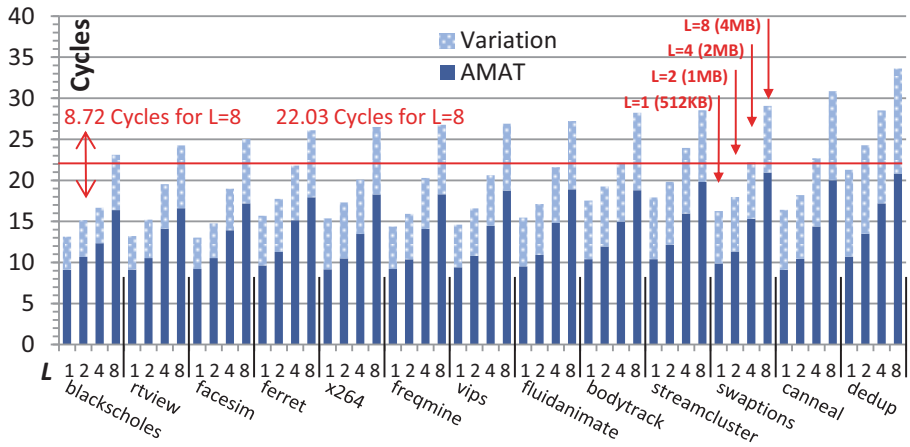


Figure 3.8: Average Memory Access Time ($AMAT$) in cycles and its variation (std. dev.) for PARSEC benchmarks, plotted for different number of stacked memory dies (L). Total L2 memory sizes changes from 512 KB to 4 MB proportional to L.

part of the benchmarks when they enter their Region of Interest (ROI).

3.6.1 Network Parameters of 3D-NUMA

For the first experiment, traces of PARSEC benchmark have been applied to the NoC Interfaces (NIs) of the baseline system (bypassing the processing clusters and their L1 caches illustrated in Figure 3.1), and Memory Access Time (MAT) has been measured for different number of stacked dies. Figure 3.8 plots Average Memory Access Time (AMAT) in cycles along with variation in access time measured as the standard deviation of the access time distribution. It can be seen that for the stack of 8 memory dies, average memory access takes 22.03 cycles with a variation of 8.72 cycles. Also an average bandwidth of 23 GB/sec. is delivered to the benchmarks, which is not illustrated in this figure.

Next, to measure the sensitivity of Memory Access Time to requested bandwidth, uniform random traffic with random command types (Load/Store of 1, 2, ..., 64 Bytes) is injected to the baseline network, and packet inter-arrival time is changed based on $Random[0, T]$ where T is plotted on the X-axis in Figure 3.9. When T=0, full-bandwidth is requested at the NoC Interfaces, and as T increases, requested bandwidth decreases. Figure 3.9.a plots AMAT for different number of stacked memory dies, and Figure 3.9.b plots delivered bandwidth of the network normalized to its upper bound calculated as $N \times W \times F = 64$GB/sec., where N is the number of NIs, W is width of

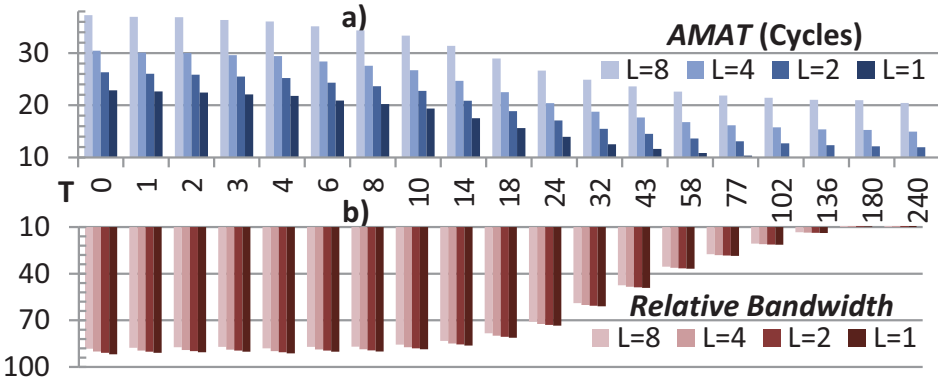


Figure 3.9: Effect of requested bandwidth on Average Memory Access Time ($AMAT$) (a) and delivered bandwidth (b) (normalized to the ideal case: 64 GB/sec.), where uniform random traffic with packet inter-arrival time of $Random[0, T]$ is applied.

the data bus in bytes, and F is the operating frequency. As can be seen, when full-bandwidth is requested, the network is able to deliver 88.2% of the bandwidth upper bound with an AMAT of 37.2 cycles, for a stack of 8 memory dies.

Equation (3.1), shows different contributors to Memory Access Time (MAT) of a *Load* command in cycles. t_{NST} represents network stall time; t_{QF}, t_{RF}, and t_{SF} are wait times in request, response, and response selector FIFOs (inside RB), respectively; t_{RE} is the wait time for Read Buffer to have an empty location; t_{ARB} is the number of cycles to wait before winning the arbitration; t_{JF1} and t_{JF2} represent the wait time in FIFOs inside the Join module (See Figure 3.2.a); and lastly, N_L is the index of the target memory die for this command.

$$MAT = t_{NST} + \{t_{QF} + t_{RF} + t_{SF}\} + \{t_{RE} + t_{ARB}\} + N_L \times \{2 + t_{JF2}\} + \{t_{JF1} - t_{JF2}\} + 3 \quad (3.1)$$

If we assume that there are no other in-flight packets, for a *Load8Bytes* command targeted to the first memory die, we can obtain a lower bound of 10 Cycles for Equation (3.1). As the packet inter-arrival time grows to infinity in Figure 3.9, AMAT for L=1 approaches this number. Moreover, for an evenly distributed traffic among all 8 memory dies, AMAT can be estimated from the formula as 20.5 Cycles. Similarly, AMAT for L=8 in Figure 3.9 approaches this number for large values of inter-arrival time. Referring back to the experiments with typical application workload (Figure 3.8), we note that the network delivers the requested bandwidth with less than

10% increase in latency with respect to an ideal banking conflict-free condition. These experiments confirm that 3D-NUMA delivers consistent performance results suitable for an L2 scratchpad memory. Next, we compare it with the state-of-the-art bank-level interleaved memories and show its benefits.

3.6.2 3D-NUMA vs Memory Banks Attached to NoC

One question which needs to be answered is that, if we remove the relatively complex wirings and arbitration trees of 3D-NUMA and attach the memory pipeline directly to the NoC through the NIs, do we gain any benefit? In fact, we have to justify that 3D-NUMA performs better than a traditional bank-level-interleaved memory directly attached to the NoC through its multiple ports [89][82]. For this purpose, we have modeled a high performance NoC based on [79] using high-level constructs in SystemVerilog. The linear array NoCs shown in Figure 3.10a,b have a clock frequency of 3.6 GHz, with enough flit size to fit a *Load8Bytes* command, and with operation rate of 1 Flit/Cycle. NoC Link latency is considered to be 1 Cycle. To maintain deadlock free communication, multiple virtual channels are implemented, and the buffer size per each virtual channel inside each NoC switch has been considered to be 32 flits. On the other hand, 3D-NUMA memory system operates at the frequency of 500 MHz.

Two scenarios are compared in this experiment: the first one consists of the complete 3D-NUMA IP connected to 16 traffic generators (TG) using one layer of the mentioned NoC switches (Figure 3.10.a). While in the second scenario (Figure 3.10.b), the arbitration trees have been removed and memories are treated as bank-level-interleaved (Memory pipeline operates at the same frequency of 500 MHz). Therefore, accesses to remote banks are routed through the NoC, and the switches should handle more pressure. Also multi-word accesses to one bank are serialized in the memory system. All other parameters are common between the two scenarios. Figure 3.10.c compares the total execution time for serving all memory accesses of a fixed number of traces from the ROI region of PARSEC 2.1 benchmarks between the two scenarios. In this experiment 3D-NUMA reduces total execution time by an average of 34%. This is firstly because the word-level interleaved organization allows for parallel access to memory banks (see next subsection for comparison with bank-level interleaving), and secondly, the arbitration trees forming a cross-bar switch reduce the traffic and congestion in the NoC switches.

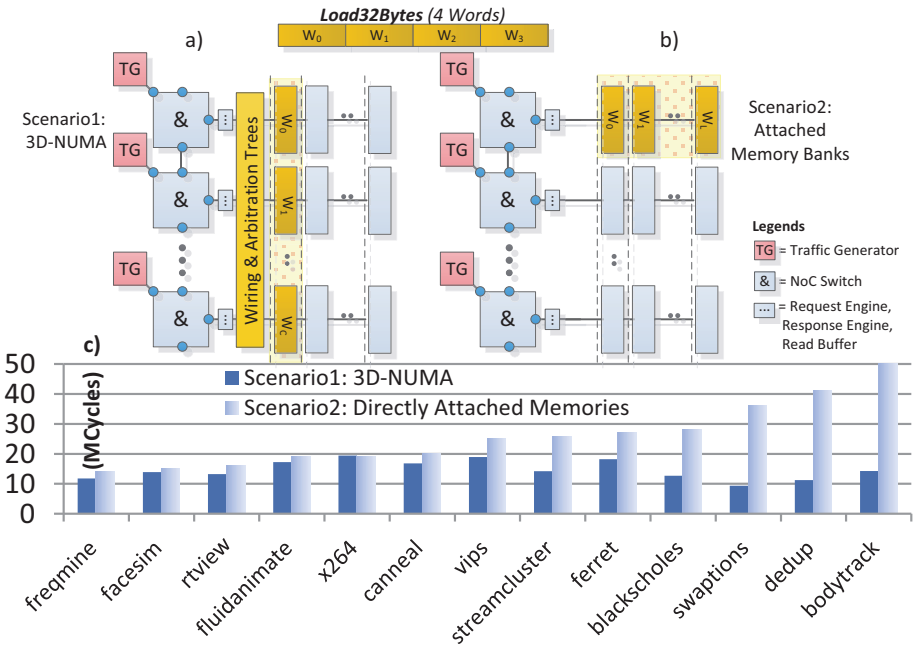


Figure 3.10: Comparison of the execution time (MCycles) between Scenario 1: 3D-NUMA (a), and Scenario 2: memory banks directly attached to NIs (b), for different PARSEC benchmarks (c).

3.6.3 Effect of Memory Interleaving

Next, we study the effect of memory interleaving on different network parameters in 3D-NUMA. Bank Level Interleaving (BLI) is the most commonly used method in memory systems [89][82][102], while 3D-NUMA has been designed in a Word Level Interleaved (WLI) fashion. Two scenarios are created for this purpose. WLI: the baseline 3D-NUMA with word-level-interleaving across memory cones (Figure 3.11.a), and BLI: a modified version of 3D-NUMA which is bank-level interleaved (Figure 3.11.b), therefore, multi-word requests are serialized in it. Figure 3.11.c compares the two scenarios for four parameters (under PARSEC benchmark traces): total execution time, Average Memory Access Time (AMAT), variation in memory access time (measured as standard deviation), and finally, Total Network Stall Time (TNST) which shows that amount of time that the request packets should wait before being granted into the memory pipeline. As can be seen, in all parameters, Word-Level-Interleaving (WLI) performs better than Bank Level Interleaving (BLI). Another experiment is performed to show this difference more clearly. Uniform random traffic is injected into both scenarios and the relative values of WLI to BLI for all four parameters are plotted in Figure 3.12. The value T on the X-axis is used to change the packet inter-arrival time

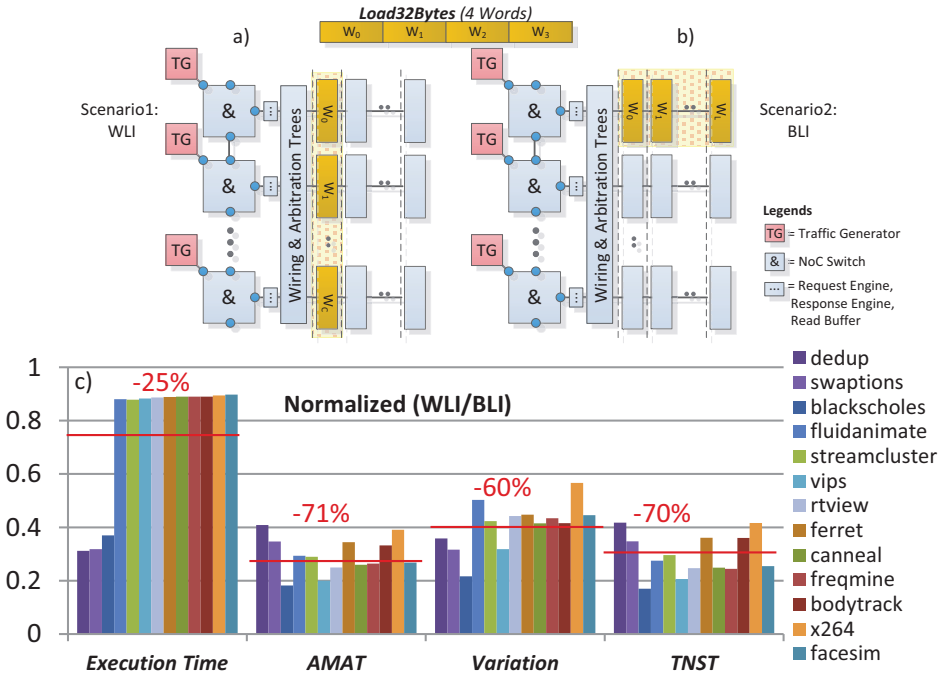


Figure 3.11: Comparison of two scenarios for 3D-NUMA: Word-level-interleaving [WLI] (a) and Bank-level-interleaving [BLI] (b); using PARSEC benchmarks (c) (Values of WLI are normalized to BLI)

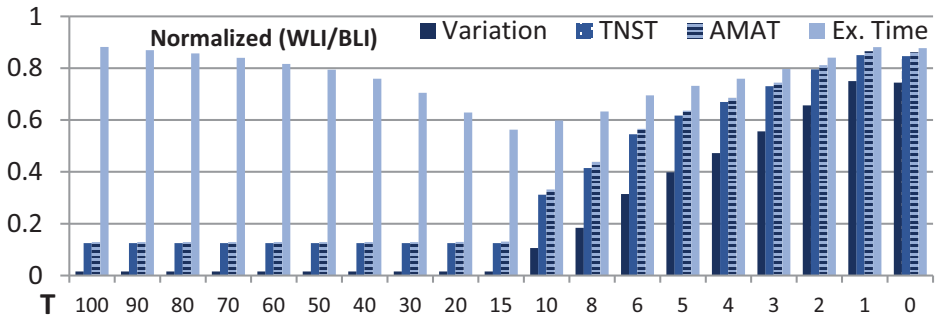


Figure 3.12: Comparison of two scenarios for 3D-NUMA under uniform random traffic with packet inter-arrival time of $Random[0, T]$ (Values of WLI are normalized to BLI).

in cycles ($Random[0, T]$). It can be seen from this figure that in all cases, execution time of WLI is at least 12.5% lower than BLI. Moreover, as inter-arrival time between packets increases, contentions in WLI reduce, therefore the benefit of using WLI improves, until at some point AMAT, TNST, and Variation stop improving any further. While, relative execution time of WLI to BLI increases afterwards because it will be dominated by the empty spaces between the request packets.

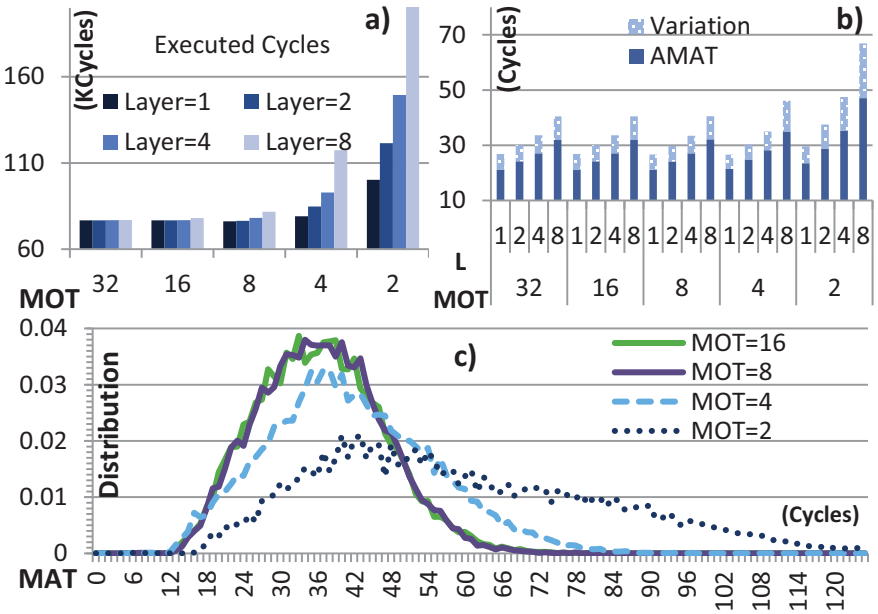


Figure 3.13: Effect of Maximum Outstanding Transactions (MOT) on execution time ($KCycles$) (a), Average Memory Access Time ($AMAT$), and access time variation, in cycles (b). Effect of MOT on the distribution of Memory Access Time (MAT) in cycles (c), full-bandwidth uniform random traffic is applied.

In addition to performance improvement, WLI offers better scalability and allows parallel serving of multi-word requests. While in BLI designs, such operation would require a change in the width of the data bus (W) and incurs a lot of complexity to support parallel serving of requests with different sizes.

3.6.4 Effect of Maximum Outstanding Transactions

As we explained before, modular design of 3D-NUMA and its highly pipelined nature result in non-uniform memory access times. In order to compensate for this behaviour, multiple outstanding transactions (up to MOT) are supported in 3D-NUMA. In this experiment, we study the effect of this critical parameter on total execution time, Average Memory Access Time (AMAT), and variation in the access time (std. dev.), for different number of stacked memory dies in 3D-NUMA. Full bandwidth uniform-random traffic with random command types is injected at the NIs. Figure 3.13 illustrates the results. It can be seen that increasing MOT reduces the total execution time of an 8-Layer memory stack down to a 1-Layer memory stack. Also, AMAT and Variation decrease when the network supports more number of in flight transactions. This is

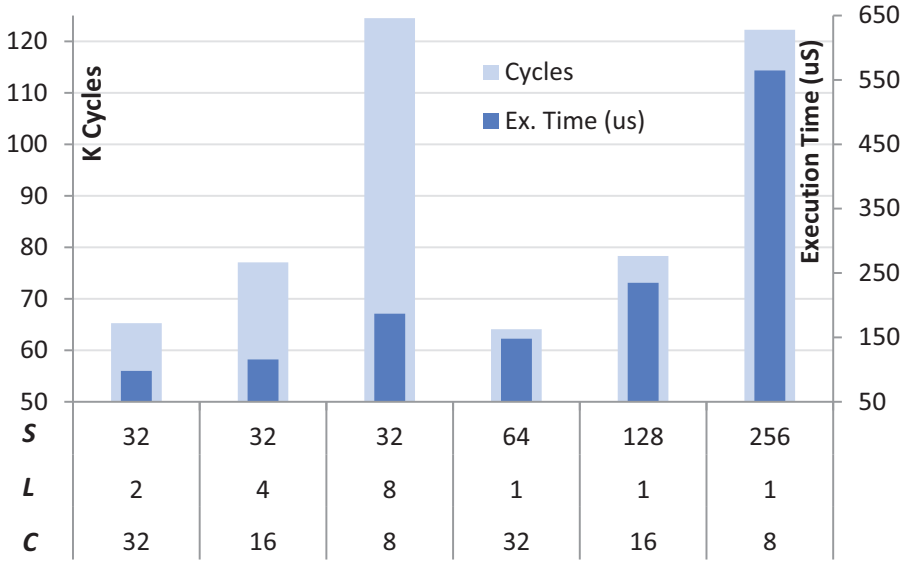


Figure 3.14: Total executed cycles and execution time (μS) for different configurations leading to the same total memory size ($S \times L \times C = 2MB$), where uniform random traffic is applied

because increasing MOT allows more in-flight requests, rather than waiting for previously issued ones to complete. Therefore, bandwidth is utilized more efficiently and memory access time is amortized. Similar techniques are usually utilized in advanced DMA engines and DDR memories to hide access latency. The NUMA behaviour and the effect of MOT on it can be further shown by plotting the distribution of the memory access time for 8 stacked memory dies in Figure 3.13.c, where the same random traffic is applied. As can be seen, increasing MOT up to 8 improves the access time mean and variance while MOT=16 makes only a slight improvement.

3.6.5 Different Configurations with Equal Memory Size

To have a better insight about different design alternatives, under random traffic execution time (μS) and total number of execution cycles of 3D-NUMA have been plotted in Figure 3.14 for several different configurations that lead to the same total memory size (i.e. $S \times L \times C = 2$ MB). Clock period has been scaled based on the synthesis results. As Figure 3.14 illustrates, the main factor which affects the clock period is the size of the memory banks (S), while increasing the number of memory cones (C) leads only to a slight increase in it. The best configuration in terms of total execution time is with (S, L, C) equal to (32, 2, 32). This is because the clock period is already limited by memory access time, and changing C up to 32 does not affect it any further. How-

ever, in terms of area, this configuration has the highest overhead, because the size of the *Read Buffers* and the number of *Arbitration Trees* grow with C. The configuration (32, 8, 8) offers reasonable execution time, low area overhead, and more importantly better scalability and modularity, while execution cycles in this configuration is more than the others, and this may translate into higher dynamic power consumption (see the next section for detailed power consumption results and the techniques to reduce it).

One last point to mention is that 3D-NUMA features similar pre-P&R latency and clock frequency in comparison with its 2D version. This is due to the highly pipelined nature of the proposed architecture, which in fact is also well suited for L2 implementation and provides several opportunities such as scalability, ease of timing closure and clock distribution. However, by going vertical, wire length, routing complexity, and synthesis effort are reduced, while in the flat design several buffers will be inserted by the synthesis tool in the long wires, and larger channels should be allocated to avoid routing congestion. This problem intensifies in the 2D planar designs as the size of the memory increases.

3.7 Power and Temperature Analysis

This section presents results related to power consumption and temperature, and proposes solutions to address related issues.

3.7.1 Power Analysis

Power consumption has been analysed with Synopsys Primetime in the typical corner case (TT) at $25°C$, with the switching activity recorded from ModelSim in Value Change Dump (VCD) format. For the first experiment, full-bandwidth random traffic with a packet inter-arrival time of $Random[0, T]$ (Cycles) is applied to the post-layout model of 3D-NUMA. Also, automatic clock gating has been enabled during the synthesis (see the next experiment for a comparison of the clock gating schemes).

Figure 3.15.a,b illustrate the power consumption in the logic die and memory die MD0, broken into memory macros, registers, combinational logic, and clock network. The largest contributors to the power consumption in LD are the memory macros utilized in the read buffers. Moreover, the power consumption in MD0 does not decrease down to zero as packet inter-arrival time increases, and a residue of about $20mW$

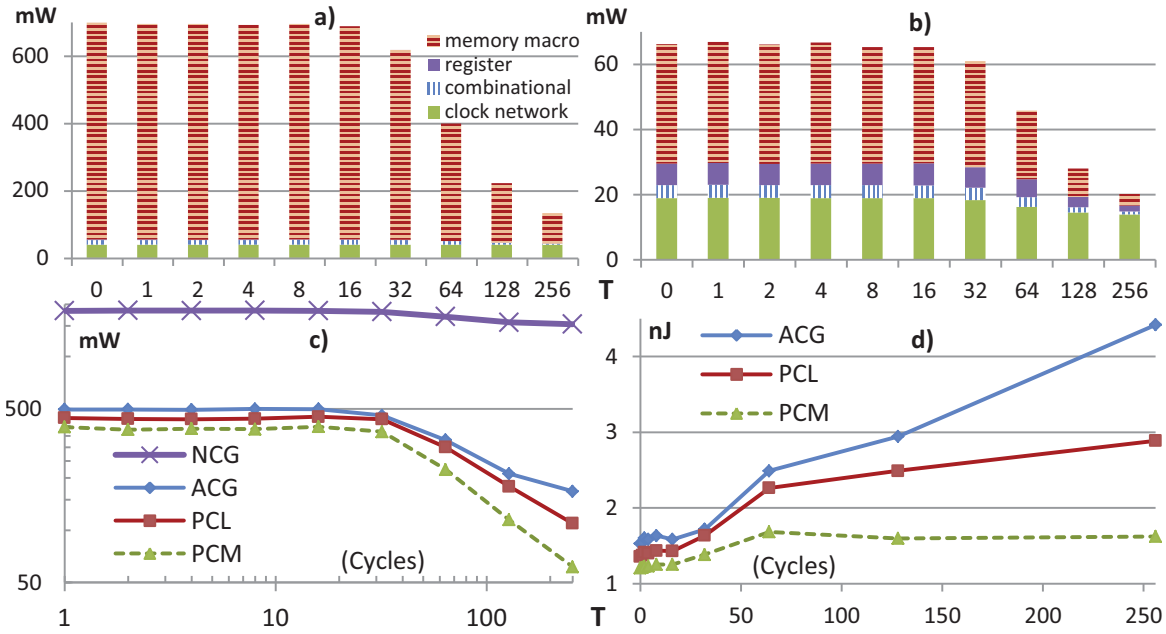


Figure 3.15: Power consumption breakdown in the logic die (a), and in memory die MD0 with automatic clock gating (b), comparison of power in the stack of 8 memory dies for four different clocking strategies (c), energy/transaction (nJ) compared in the same experiment (d). In all experiments, uniform random traffic has been applied with a packet inter-arrival time of *Random[0,T]*).

remains even without existence of any useful transactions. This power is mostly consumed in the clock tree and clock gating elements, and there is no easy way to remove it automatically.

In order to further reduce this power consumption in the memory stack, we propose Patient Clock (PC), an architectural clock gating mechanism which clocks the memory dies only if there is a pending or in-flight request. This is done in two simple ways: Patient Clock in Logic die (PCL) and Patient Clock in Memory dies (PCM). In PCL, the logic die gates the input clock to the memory stack, so all memory arrays which form a *cone* receive the same gated clock and are enabled together. While in PCM, the clock to each individual memory array is gated based on requests from lower dies and the responses from upper dies (See PC in Figure 3.2.a). To implement patient clock, a very simple up/down counter is utilized which counts up on every request and down on every response. As long as this counter is equal to zero, clock is gated. Otherwise, clock is activated to serve the in-flight requests and responses. It should be noted that patient clock is completely transparent to the 3D interface, and the gated

clock generated in each memory die is sent to the next memory die as its main clock. Figure 3.15.c compares the post-layout power consumption among the four different clocking strategies: No Clock-Gating (NCG), Automatic Clock-Gating (ACG) during the synthesis phase, Patient Clock in Logic die (PCL), and Patient Clock in Memory dies (PCM) (described previously). Uniform random traffic with a packet inter-arrival time of $Random[0, T]$ (Cycles) has been applied to the NoC interfaces. It can be seen that the original design with no clock gating (NCG) performs poorly in all cases (about 3.7× more power consumption than ACG), therefore, in all other experiments (including temperature analysis) ACG mechanism is utilized. In addition, power consumption in all cases drops rapidly after T exceeds almost 30 cycles. Therefore, the clock gating methods are suitable only when long pauses exist among the packets. This seems achievable in L2 memory, since L1 caches filter out the requests from PEs and turn them into large packets of refill/write-back (see Figure 3.16 for real benchmarking results). Furthermore, the pipelined nature of 3D-NUMA results in lower signal activity in the upper memory dies. This way the benefit of the gated clocks increases in the upper memory dies. Next, Figure 3.15.d compares *energy/transaction* among the clock gating methods in the previous experiment. It can be seen that PCM is the most energy efficient method among the three, and its energy consumption is almost independent from the inter-arrival time among the packets.

In order to see how these methods perform under realistic loads, experiments are repeated with PARSEC benchmarks. Figure 3.16.a illustrates the percentage of reduction in switching activity in the stack of 8 memory dies for each benchmark after PCL is utilized. Figure 3.16.b illustrates this reduction in each memory die with PCM. The benchmarks react differently to different clock gating methods, nevertheless, an average reduction of 21% for PCL and 44% for PCM are observable. Total power consumption in the stack of 8 memory dies with different clock gating methods is compared in Figure 3.16.c. Here, a power reduction of 18.3% for the PCL method and 38.6% for PCM is observable (the area increase in the memory dies due to addition of PCM was less than 1%).

One last point to mention is that, even though IR-drops and voltage droops are the main limiting factors in power delivery to a 3D stack [81][103], and different algorithms for estimation of the optimal place/size/count of TSVs and thermal diodes have been proposed already [81][104], in the PIMD configuration utilized in our experiments (illustrated in Figure 3.18 and described in the next section), they are not much of a concern. This is because in this configuration the TSV interposer is placed

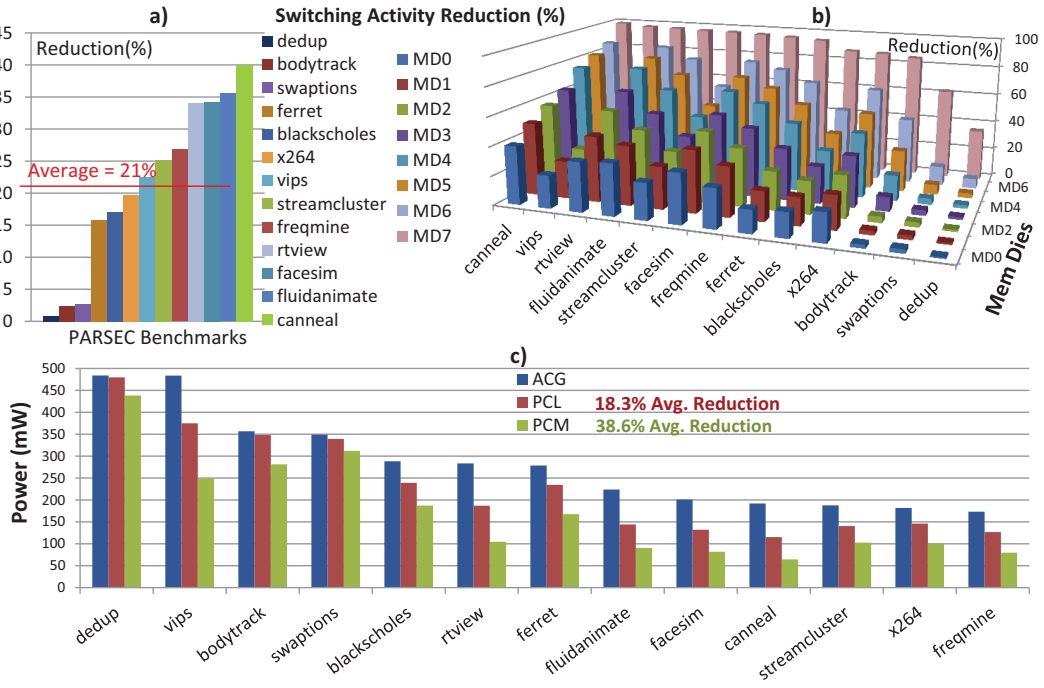


Figure 3.16: Reduction in switching activity: in a stack of 8 memory die with *PCL* (a), in memory dies with *PCM* (b), total consumed power in the stack of 8 memory dies compared between the clock gating methods (c)

between the PE die and the memory dies, and is responsible for power delivery to them, independently. Moreover, the power consumption in the memory stack is quite low ($1200mW$ in the whole stack of 8 memory dies and one logic die, under worst case load), therefore resistive drop in the power distribution network of 3D-NUMA is expected to be small. In addition, in comparison with the 2D counterpart, 3D-NUMA does not necessarily improve or worsen power consumption. Because on one hand more power is consumed in the 3D interfaces, while on the other hand, clock trees and routing channels, which are important contributors to power consumption, are simplified compared to the planar version.

3.7.2 Thermal Analysis

For thermal simulation, 3D-ICE (version 2.2.5) [105] has been utilized. Floorplan is extracted from Cadence SoC Encounter and fed into 3D-ICE using automated scripts. For generation of the power traces, Primetime is executed in small epochs of $1ms$, and again, resulting log files are analysed automatically to generate power traces for 3D-

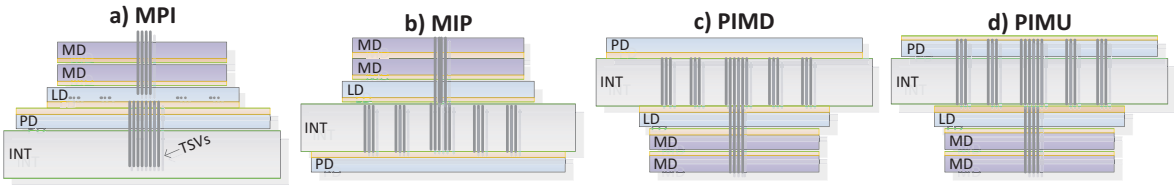


Figure 3.17: Four configurations for 3D-stacking of 3D-NUMA: Memory-Processor-Interposer (MPI) (a), Memory-Interposer-Processor (MIP) (b), Processor-Interpose-Memory faced Down (PIMD) (c), and Processor-Interposer-Memory faced Up (PIMU) (d)

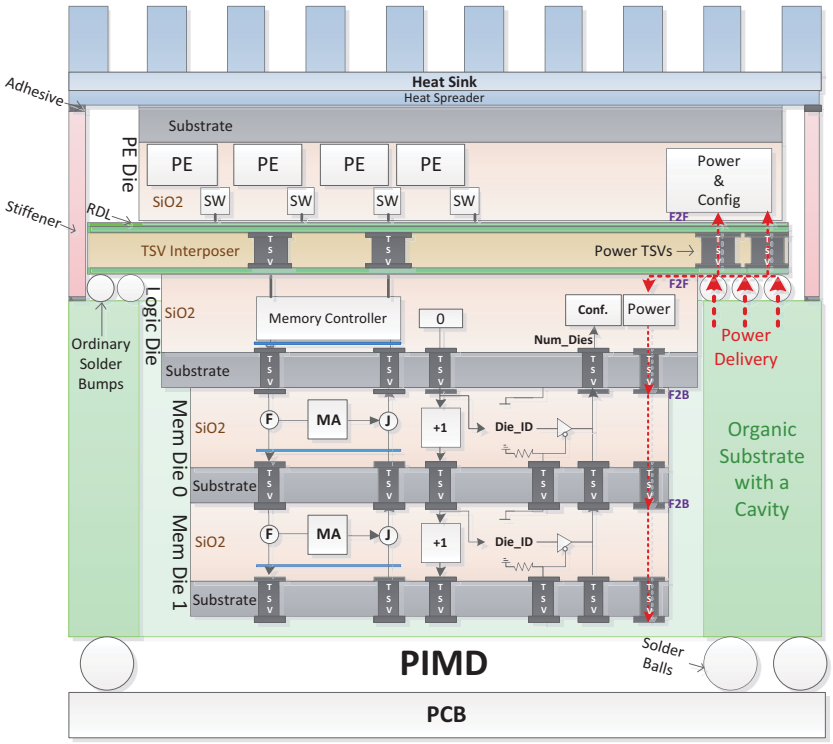


Figure 3.18: Cross section of the PIMD configuration

ICE. Four configurations have been compared from thermal point of view: MPI, MIP, PIMD, and PIMU (see Figure 3.17.a,b,c,d). In MPI the whole 3D-NUMA memory stack is placed over a PE die (PD) on the top of an interposer. The PE die (PD) and the logic die (LD) are stacked face-to-face, while the memory dies (MD) use face-to-back stacking. In the other three configurations however, the interposer separates the PE die and the memory stack. In MIP, the PE die is placed under the interposer while the memory stack is placed on its top. PIMD can be seen as a specular version of MIP, and PIMU is similar to the latter configuration except that its PE die is directed up towards the heat-sink. PE die (PD) consists of 16 STxP70 processing elements [44] connected to a mesh NoC with 16 switches. 3D-NUMA is attached to this NoC through its NIs.

For thermal analysis, ambient temperature is assumed to be $300°K$. A copper heat-sink ($2mm \times 16mm^2$) with a heat transfer coefficient (HTC) of $10^{-7}Watt/(\mu m^2.K)$, and a ceramic heat spreader ($1mm \times 5mm^2$) with a thermal conductivity of $39 \times 10^{-6}W/(\mu m.K)$ are used on the top of the stack. On the bottom side, a PCB layer is used with a thickness of $1mm$, thermal conductivity of $2.25 \times 10^{-6}W/(\mu m.K)$, and volumetric heat capacity of $2.17 \times 10^{-12}J/(m^3.K)$. All other sides of the stack are considered as "adiabatic walls" in the 3D-ICE simulator. Thickness of the silicon and BEOL materials are adopted from STM Bulk CMOS-28nm technology. Thickness of the interposer is set to $100\mu m$ and the stacked dies are thinned down to $25\mu m$. Heat transfer through the TSVs has been modeled by modifying the thermal conductivity of each layer in the stack based on the number of the TSVs passing through it, material of the TSVs, and their pitch. For this purpose pinfins in 3D-ICE have been exploited to model Copper TSVs with a diameter of $5\mu m$, then for different TSV pitch values, thermal conductivity has been plotted for each material. Finally, the obtained values have been scaled based on the relative area occupied by the TSVs in each individual layer in the 3D stack. This approximation method has been adopted from [9], and it has been necessary since 3D-ICE does not support modeling TSVs. More accurate models and analysis of the analysis of the effect of copper-filled TSVs on heat transfer has been performed in [106] and [107]. Utilization of thermal vias [108], liquid micro-channel cooling [109] and thermal isolation technologies [110] can improve thermal results in hot stacks. Nevertheless, here we utilize a passive heat-sink exposed to the ambient temperature, and to be conservative, we utilize ACG as our clocking scheme.

For the first experiment, the four configurations in Figure 3.17 are cooled down from an initial temperature of $400°K$. As can be seen in Figure 3.19.a, PIMD cools

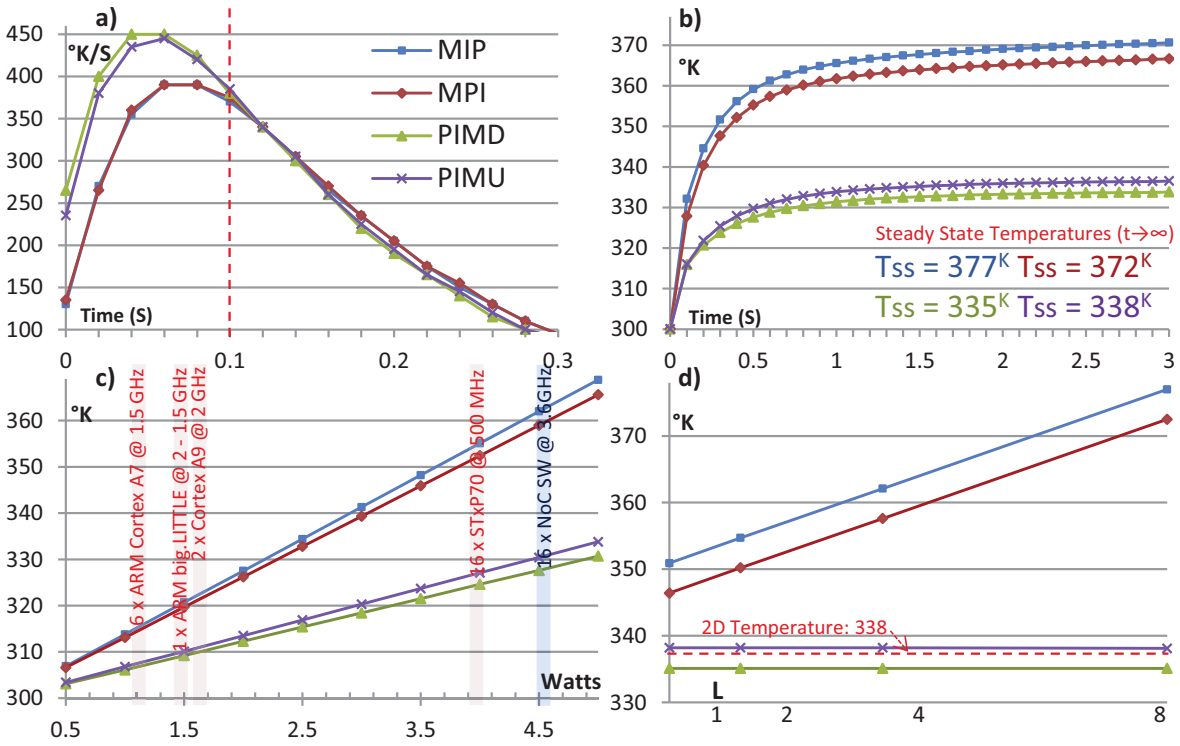


Figure 3.19: Temperature cool down rate (° K/Seconds) of the PE Die in the four configurations (a), and its transient temperature (° K) increase due to constant power consumption in the PE die [T_{SS}: steady state temperature] (b), Effect of power consumption (W) in the PE die on its temperature (° K)(c), PE die temperature versus the number of stacked memory dies [L] (d).

down faster than the three other configurations, with an average cool-down rate of $395°K/Seconds$ in the first $100ms$, while MIP has a rate of $317°K/Seconds$. This can be important when dynamic temperature management techniques are utilized to manage hotspots. In the second experiment, the processing elements on the PE die consume a constant power of $250mW$, while the NoC switches consume $100mW$ each. Figure 3.19.b depicts the transient temperature change in the center of the PE die in the four configurations along with its steady-state temperature. Here a significant temperature difference of 42° is observable between the MIP and PIMD configurations. Next, the average power consumption in the whole PE die each is changed linearly from $500mW$ to $5W$. Figure 3.19.c plots the maximum temperature in this die in the four 3D configurations. Moreover an estimated power consumption of four PE configurations along with the NoC switches utilized in our experiments have been plotted in this figure to better illustrate 3D-NUMA's thermal performance. 16 STxP70 PEs running

at 500MHz, 6 ARM Cortex A7 cores at 1.5GHz, One ARM big.LITTLE with its cores running at 1.5GHz and 2GHz, and lastly, 2 Cortex A9 cores running at 2GHz (All configurations fit in the same area provided by the PE die) [111]. As power consumption increases it can be seen that, temperature increases more rapidly in MIP and MPI compared to the two other cases. Lastly, the number of stacked memory dies is changed from 1 to 8 while each PE consumes $250mW$, and the switches consume $100mW$ each. The maximum temperature in the PE die is plotted in Figure 3.19.d. Also in this experiment, an estimated floorplan of the 2D counterpart of 3D-NUMA is simulated in 3D-ICE consuming similar power profiles, and the maximum temperature of the 2D die is plotted using a dotted line. As this figure illustrates, PIMD and PIMU scale well with the number of stacked dies, and the temperature of the PE die is almost independent from the number of the stacked dies in them. While in MPI and MIP, temperatures increase linearly with the number of the stacked memory dies. Furthermore, it can be seen that PIMD and PIMU show similar thermal results in comparison with the 2D version.

These experiments demonstrated that placing dies over a high-power die such as PD is not an optimal solution for 3D-stacking from the thermal point of view, especially when the number of stacked dies on its top is high. Moreover, PIMD seems to be a better configuration than PIMU, even though in PIMU the PE die is directed towards the heat-sink. The reason for this phenomenon is that the thermal conductivity of silicon ($149 \times 10^{-6} W/\mu m.K$) is higher than the BEOL materials ($2.2 \times 10^{-6} W/\mu m.K$). Therefore, heat transfer towards the heat-sink is facilitated when the upper die is faced down. For the same reason, the thermal performance of the PIMD configuration is slightly better than the 2D design, while TSVs also contribute to this fact (See Figure 3.19.d). It should be reminded that the materials utilized in the stack also have a significant effect on the temperature. For example, with open-air as heat sink [Heat Transfer Coefficient (HTC) of about 1e-10 $Watt/(\mu m^2.K)$], all temperatures increase to over $3000°K$, and there will be no significant difference among them, while proper heat-sinks can increase HTC to about 1e-7 $Watt/(\mu m^2.K)$ and significantly improve the heat transfer.

In the final experiment, full-bandwidth random traffic is applied to 3D-NUMA, and gathered power traces are fed into the stack in 3D-ICE. For PEs, same as before, a constant power consumption of $250mW$ is assumed (the effect of PARSEC benchmarks on power consumption of the PEs has not been modeled), and for each NoC switch $100mW$ is assumed. Figure 3.20.a,b,c,d compares the temperature map in the four con-

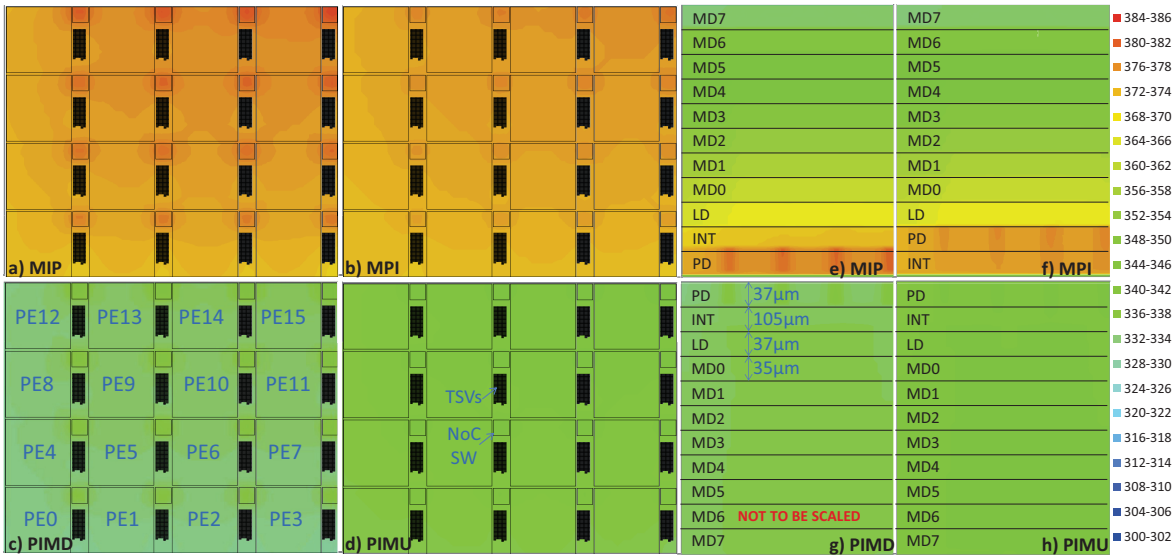


Figure 3.20: Final temperature map of the PE die in the four configurations: MIP (a), MPI (b), PIMD (c), PIMU (d), cross-sections of the whole stacks (e,f,g,h).

figurations in the end of a long simulation. Again, a maximum temperature difference of over $40°K$ in the PE die is observable between MIP and PIMD. Figure 3.20.e,f,g,h illustrate this situation better, by showing their cross-sections. In all of the experiments (including all PARSEC benchmarks) the increase in temperature due to the activity in the memory system was less than 5°. This result is not surprising because 3D-NUMA is composed of SRAM-based memories and a small percentage of logic. Therefore, it consumes much less power than the PEs.

3.8 Packaging and Power Delivery

For power delivery to 3D-NUMA several factors should be considered and different analyses such as IR-Drop, and voltage droops in 3D power distribution networks should be performed [81][103]. However an estimation of the number of power TSVS required can be performed as follows: The maximum power consumption in the Logic Die (LD) and the stack of 8 Memory Dies (MDs) are $700mW$ and $500mW$, respectively. These values have been obtained from Primetime with ACG clocking scheme. Moreover, assuming 16 STxP70 PEs and 16 NoC switches, PE Die (PD) consumes a maximum power of $5.6W$. Considering an operating voltage of 1V, and that each TSV can deliver a maximum current of 20mA [112], the number of TSVs required to deliver

Table 3.1: Number of power TSVs required for each die, and the total count for manufacturing one instance of each

Configuration	Power TSVs in each die				Power TSVs
	MD	LD	PD	INT	
MPI	25	25	340	340	730
MIP	25	25	0	60	110
PIMD	25	25	0	280	330
PIMU	25	25	280	280	610

power to the PD, LD, and MD are 280, 35, and 25, respectively. Now having in mind that, each die is responsible for power delivery to the dies stacked over it, and that face-to-face interfaces do not require TSVs (see Figure 3.18), we can obtain the number of power TSVs in each die and the total number of power TSVs required for manufacturing one instance of each in Table 3.1. It can be seen that, MIP and PIMD have the lowest number of TSVs among all, while because of PIMD's pleasant thermal results we choose it as our target configuration. It should be noted that, if we use one ground TSV per each power TSV, the numbers in Table 3.1 will double. Also, to improve signal integrity more power and ground TSVs may be required. Lastly, by use of new bonding techniques such as multi-tier wire-bonding [36], we may be able to improve power delivery and reduce the number of TSVs, as well. Since a $1mm$ copper wire-bond with a typical diameter of $15\mu m$ can deliver a maximum current of $600mA$ [113]. We should remind that wire-bonding to the PE die is not possible in the PIMD configuration and other configurations should be adopted for this purpose.

3.9 Manufacturing Yield and Cost

3D-NUMA can provide a good possibility for manufacturing yield improvement in comparison with its 2D counterpart. Equation 3.2 provides an estimation of the yield of the 3D stack (assuming that the sources of die defects and TSV failures are independent), when Known Good Die (KGD) testing mechanisms [114] are utilized and failed dies are discarded beforehand. Die yield (Y_{die}) can be estimated using Equation 3.3 with a negative binomial model, based on wafer yield (Y_{wafer}), die area (A_{die}), and the defect density of the wafer (D_0) [115]. Moreover, since a good chip stack requires all TSVs to be successfully bonded, the bonding yield (Y_{TSV}) can be estimated from Equation 3.4, where F stands for TSV failure rate, and N_{TSV} is the total number of TSVs in

the stack [98]. Now obtaining TSV failure rate from the state-of-the-art technologies [22][116][117], and assuming that the CMOS-28nm process technology is in the fourth learning cycle of its yield curve ($D_0 \simeq 1.5cm^{-2}$[118]), we can plot the 3D manufacturing yield of the PIMD configuration for different TSV failure rates in Figure 3.21. The TSV repair mechanism utilized in 3D-NUMA is able to fix up to 5 TSVs with a probability of 95% [98], and the probability that exactly n TSVs fail can be obtained from Equation 3.5 [98]. Now if the method is able to fix n TSVs, for failures less than this number the stack is usable, therefore the bonding yield can be updated to Y^*_{TSV} as in Equation 3.6.

$$Y_{stack} = Y_{die} \times Y_{TSV} \tag{3.2}$$

$$Y_{die} = Y_{wafer} \times [1 + D_0 \times A_{die}/\alpha]^{-\alpha} \tag{3.3}$$

$$Y_{TSV} = (1 - F)^{N_{TSV}} \tag{3.4}$$

$$P_{failed=n} = C(N_{TSV}, n) \times F^n \times (1 - F)^{N_{TSV}-n} \tag{3.5}$$

$$Y^*_{TSV} = \sum\nolimits_{i=0}^{n} P_{failed=i} \tag{3.6}$$

Figure 3.21 plots the manufacturing yield for 3D-NUMA without TSV redundancy along with the repair scheme up to 5 TSV fixes. The 2D yield has been estimated similarly using Equation 3.3. As can be seen, for a more mature TSV technology by Honda Research Institute (HRI), even without any repair mechanism, 22% yield improvement can be obtained in 3D-NUMA for stacking of 4MB of scratchpad memory as 8 dies, compared to the 2D flat counterpart.

Another point to mention is that, 3D-NUMA memory IP has certain features which benefit from the cost optimization opportunities provided by 3D-Integration. The fact that logic and memory elements have been separated into different dies, allows for different technological optimizations for logic and memories as well as reduction in the number of metal layers in the memory dies (because of lower routing complexity and more regular patterns). In addition, certain boot-time configuration circuits allow memory dies to have completely identical layouts (see Figure 3.18). Therefore, system integrators are allowed to stack multiple memory dies and create arbitrary L2 memory sizes through different height stacks with identical dies, without the need for new masks for dies at different levels in the stack.

We developed a generic cost model for the CMOS 28nm technology based on the public information about the cost of masks and lithography. The cost model of masks and lithography is described in [119] for the 90nm technology node. Later, it has been

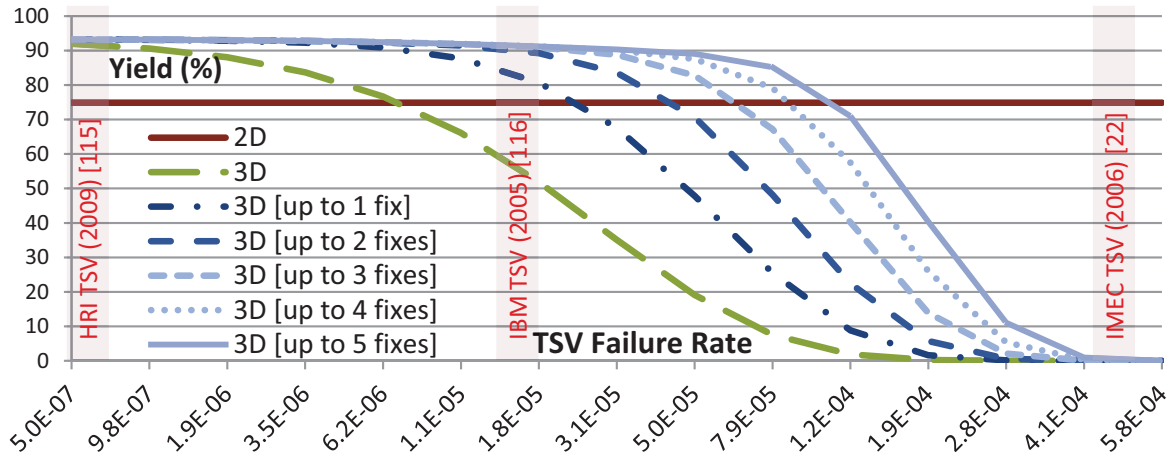


Figure 3.21: Manufacturing yield versus TSV failure rate compared between 3D-NUMA with no TSV redundancy, TSV Repair Scheme (Up to 5 Fixes), and the 2D counterpart

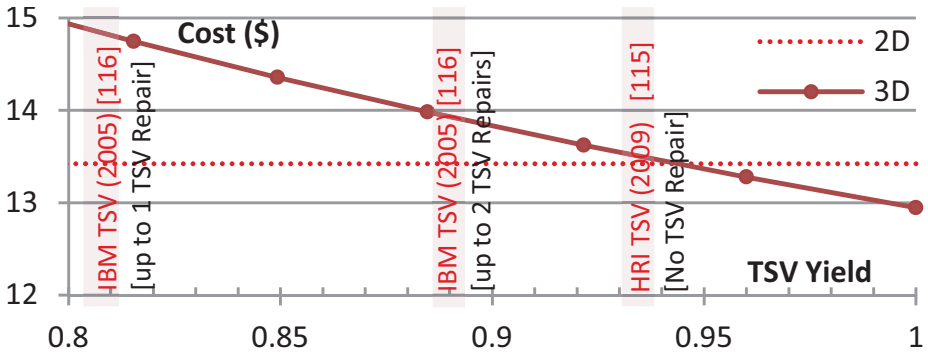


Figure 3.22: Overall manufacturing cost of one PIMD stack compared its 2D counterpart

extended to more recent technology nodes in [120]. The manufacturing yield model is described in [115], and the TSV yield has been modeled as described previously. Packaging costs and special treatments for 3D integration have not been modeled. This is because this technology has not been industrialized yet and the supply chain for low-cost and high-volume production of 3D-ICs still does not exist. Figure 3.22 compares the overall manufacturing cost of one stack (with 4MB of memory stacked as 8 dies) with its 2D counterpart, for a volume production of 500,000.

It can be seen that, 3D integration has more cost reduction potentials compared to 2D as the TSV technologies become more mature. However, the cost reduction plotted in this figure is not so high because of the small area of the memory dies in 3D-NUMA. While, with increase in the size of the stacked memories cost reduction opportunities

will increase, as well. This result is based on the assumption that the memory dies have identical layouts, therefore their non recurring costs will overlap (e.g. mask cost). Whereas, if we use memory dies with non-identical layouts, this cost will increase by a factor of 2.3 to 30.2$ (even with ideal TSVs). This is a significant cost reduction offered by 3D-NUMA over the designs with non-identical memory dies. Moreover, it is possible to use a less advanced process technology or reduced number of metal layers for the memory dies to reduce the costs even further. While this is out of the scope of this work, in the design of the memory dies we have reduced the number of metal layers from 10 to 8 to save four masks.

3.10 Summary

In this chapter, we presented a synthesizable 3D-stackable L2 memory IP component (3D-NUMA), which could be attached to a cluster-based multi-core platform through its NIs, offering high-bandwidth memory access with low average latency. 3D-NUMA allows stacking of multiple identical memory dies, supports multiple outstanding transactions, and achieves high clock frequencies due to its highly pipelined nature. We implemented 3D-NUMA with STMicroelectronics CMOS-28nm Low Power Technology and obtained a clock frequency of 500 MHz, limited by the access time of the memory arrays while its logic components could operate up to 1 GHz (up to 4 MB in 8 stacked dies with a memory density loss of 9%). Benchmark simulation results demonstrate that addition of 3D-NUMA to a multi-cluster system can lead to an average performance boost of 34%. Further experiments and estimations confirmed that 3D-NUMA is energy and power efficient, temperature friendly, and has unique features suitable for low cost manufacturing: PCM architectural clock gating mechanism was proposed to reduce power consumption by 38%. PIMD configuration was able to reduce maximum temperature by over 40° C in comparison with the conventional memory on top configurations. Lastly, 2.3× cost reduction was reported because of identical memory die layouts along with a 22% yield improvement compared to the 2D counterpart, with the state-of-the-art TSV manufacturing technologies.

In the next two chapters (chapter 4 and chapter 5), we move towards the last level in the memory hierarchy, and study the benefits provided by 3D integration, there. Since, there is already industrial evidence for 3D stacked DRAMs, we take one step further and focus on another important impact of heterogeneous 3D integration, i.e. possibility of near memory computation.

Chapter 4

Near Memory Computation in the L3 Memory Context

In chapter 2 and chapter 3 we studied the applicability of 3D integration in lower levels of the memory hierarchy. In this chapter, we investigate its benefits in the last level of the memory hierarchy (i.e. the main DRAM memory). There is no need to justify the effectiveness of 3D integration in this context, as several academic and industrial example exist, with Hybrid Memory Cube (HMC) being their most famous one. HMC is backed by several major semiconductor companies and has promised to improve bandwidth, power consumption, and density for the next-generation main memory systems. Meanwhile, heterogeneous 3-D integration provides another opportunity for revisiting near memory computation to fill the gap between the processors and memories. We take advantage of this opportunity and propose the "Smart Memory Cube (SMC)", a fully backward compatible and modular extension to the standard HMC, supporting near memory computation on its Logic Base (LoB). In this chapter we focus on the architectural implications and the required infrastructure inside HMC to support this feature. We propose a high bandwidth, low latency, and AXI-4.0 compatible interconnect for LoB to serve the huge bandwidth demand by HMC's serial links, and to provide extra bandwidth to a processor-in-memory (PIM) embedded in LoB. We also implement a novel address scrambling mechanism which allows for reducing vault/bank conflicts and robust operation in presence of pathological traffic patterns.

4.1 Motivations and Challenges

The "memory wall problem", or the speed and bandwidth disparity between processors and memory, has been a concern for the last thirty years [16]. Many researchers, since the early nineties [45], have looked into the possibility to migrate some part of computation closer to the memory systems. Unfortunately, the "processing-in-memory" research efforts in the late nineties and the first decade of the new millennium (See [45][46][17] for samples) did not lead to successful industrial platforms and products. The main reason for this lack of success was that all these works were assuming that significant amount of logic resources, needed for having processing elements close to the memory arrays, could be integrated on DRAM dies (or vice versa). This could not be achieved economically given the restrictions of DRAM processes (e.g., limited number of metal levels, slow transistors). On the other hand, integration of DRAM in logic processes has achieved some partial success, but it has always been plagued by high cost and low memory density issues [18].

Starting from 2011, this situation started to change with the appearance of heterogeneous 3D integration of logic dies and memory dies based on through-silicon-vias (TSV). TSV technology was brought to commercial maturity by memory manufacturers (DRAM and Flash) to build "memory-cubes" made of vertically stacked thinned memory dies which achieve higher capacity in packages with smaller footprint and power compared to traditional multi-chip modules. The last missing piece came in place when an industrial consortium backed by several major semiconductor companies introduced the Hybrid Memory Cube (HMC). [13] In the HMC, a memory cube is stacked on top of a logic die. The logic die at the bottom of the hybrid stack provides advanced interface functions between the memory cube on top and the rest of the computing system on the board. The main driver for the HMC has been bandwidth: the high-speed logic die is used to build fast serial IO transceivers for off-chip communication, on-chip controllers and interconnects for multiplexing the vertically stacked memory partitions (called "vaults").

In this chapter, we leverage the recent technology breakthrough represented by the HMC to revisit the possibility of near memory processing inside the cube, taking advantage of the heterogeneous 3D stacking technology. We mainly focus on the architectural implications and the required infrastructure inside HMC to support this feature. Therefore, exploiting the high internal bandwidth provided by TSVs we propose a modular and scalable solution, called the "Smart Memory Cube (SMC)". SMC

is built upon the most recent revision of the HMC specifications, and is compatible with its interface, with no changes made to the memory dies, and no new die introduced in the stack. In other words, SMC is fully backward compatible with the HMC IO interface specification, and features a high performance and extensible AXI-4.0 based interconnect on its Logic Base (LoB), carefully designed to provide high bandwidth to the external serial links, as well as plenty of extra bandwidth to any generic and AXI-compliant PIM device attached to its extension ports. Its also features a novel address scrambling mechanism for reducing vault/bank conflicts and robust operation in presence of pathological traffic patterns. Cycle accurate (CA) models for the SMC interconnect and its interfaces have been developed, and their parameters are tuned based on the available data from the literature on HMC.

Related works are discussed in section 4.2. In section 4.3 SMC and its CA model are introduced. In section 4.4 calibration of the model based on the available data on HMC is described. Experimental results are presented in section 4.5, and lastly a summary of the obtained results is given in section 4.6.

4.2 Related Works

While the advancement of processor technology has rapidly increased computational capabilities in logic processes, improvements in bandwidth and latency to off-chip memory have not kept up, and in fact DRAM process is drifting further away from the logic process. As a result, an increasing portion of time and power in computing systems is spent on data movement, especially in off-chip memory accesses [16]. A possible solution to the memory wall problem is in-memory processing. Research in this area started more than two decades ago. Computational RAM [45] using SRAMs or DRAMs coupled with processing elements close to the sense amplifiers, and Intelligent-RAM (IRAM) [46] to fill the gap between DRAM and processors, are just two examples of the efforts in this area. It was shown in [46] that in memory processing can lead to a memory bandwidth and energy efficiency improvement of 50X$\sim$100X and 2X respectively, along with a latency reduction of about 2X (for the SPEC benchmarks suite). Nevertheless, the effort for PIM dried out soon after 2000's without major commercial adopters, due to several performance, cost, and business model obstacles [17], arising from the incompatibility of DRAM process with logic.

With the recent advancements in process technology and emergence of 3D integration, the interest in near-memory computation has been renewed [47]. 3D memory

stacking has been the biggest driver for high-volume adoption of the 3D Integration technology, providing this new context for PIM research. The most outstanding examples of 3D memory stacking as substitutes for traditional DDR devices are the Hybrid Memory Cube [91], the High Bandwidth Memory (HBM) [26], and the Exascale Memory [27]. Among these, HMC offers higher flexibility by abstracting away the details of DRAM control, and providing a high-level communication mechanism over serial links. Therefore we believe that HMC is the best target for near memory computation.

Focusing on the location of the PIM device in HMC, it can be either integrated with the existing logic [16][121][53][122] or DRAM dies [123], or it can be added as a new die in the stack [124][49]. Introduction of a new layer to the stack would require redesign and a complete reanalysis of the 3D stack structure and the power distribution networks [106], affecting manufacturing yield of the stack, as well. Also, placing the PIM devices on the memory dies still suffers from the incompatibility of logic and DRAM processes [17] and the functionality and visibility of the PIM device to the address space will become extremely limited. On the other hand, placing the PIM device on the logic die, specifically behind the main interconnect in the HMC (See Figure 4.1), could lead to a modular and scalable solution, with a global visibility of the whole memory space, exploiting the large bandwidth provided by TSVs, and without any concerns about the DRAM devices. Besides, this solution is the least intrusive one to the standard HMC architecture, as it does not make any change to the 3D stack or the DRAM dies.

Address interleaving and remapping has been studied in the literature for different purposes. [125] shows enhanced address mapping strategies for improving reliability in 3D NAND Flash Memories. Also, in [126] a randomized addressing scheme has been proposed to improve the endurance of the phase-change-memories (PCM). [127] and [128] propose tuning the addressing scheme to specific applications for the sake of access conflict reduction. [127] presents the mathematical models for address interleavers to reduce memory collision in turbo decoders and [128] proposes a conflict-free memory addressing scheme for parallel Fast Fourier Transform (FFT) processors. Permutation-based address interleaving for reduction in the row-buffer conflicts has been studied in [129]. In [130], also, different addressing scheme to mitigate the performance impact of row buffer conflicts and to improve the locality are explored. The main difference between our proposal and the state-of-the-art is that, firstly, instead of tuning the address mapping mechanism to specific applications we present a general address scrambling mechanism to reduce the bank and vault conflicts in the standard HMC regardless of the address patterns of the input traffic. Moreover, as section 4.4 will describe, since

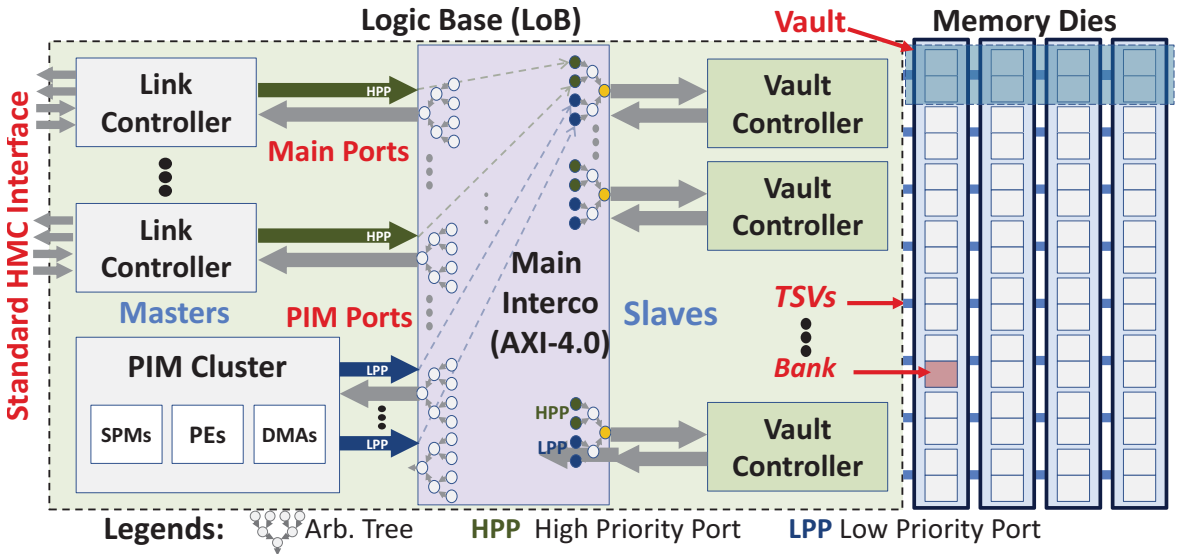


Figure 4.1: Overview of the Smart Memory Cube (SMC)

the HMC has been designed with a closed-page policy and for random traffic patterns with with very low locality, improving the row-buffer's hit rate is not relevant in it anymore. Plus, as we will describe in subsection 4.3.2, HMC's addressing scheme is limited to a small number of permutations among different portions of the input address. We will show in subsection 4.5.2 that this makes it vulnerable to specific traffic patterns of some real applications. These traffic patterns are studied in subsection 4.5.2 and it will be shown that our proposal is able to operate robustly even in their presence. Lastly, since the overhead of dynamically changing the addressing scheme is very high requiring an invalidation of all dirty pages in the main memory, we propose a simple region-based mechanism, to turn On/Off the scrambling feature for different regions.

One last point to mention is that, in the standard published by HMC consortium [13], the external interface is specified in complete details, while, the implementation details of the Logic Base (LoB), the DRAM dies, and specially the main interconnect inside the LoB have been left open. Our main contribution in this chapter is to design a high performance and low latency interconnect based on the AXI-4.0 standard to serve as the main interconnect in HMC, while providing additional bandwidth to a generic PIM device attached to it, minimizing vault and bank conflicts and ensuring that interference on the main traffic is minimum. Next section describes our proposal called the smart memory cube.

4.3 The Smart Memory Cube (SMC)

Smart Memory Cube (SMC) is an extension to HMC [13] providing the possibility of moving part of the computation into the cube. Figure 4.1 illustrates an overview of the proposed underlying architecture for Smart Memory Cube. As shown in this figure, the standard interface and the 3D structure of the HMC has been left untouched, while a generic processor-in-memory (PIM) cluster has been added to the Logic Base (LoB) behind the global interconnect. It is important to guarantee that the interference caused by PIM's traffic on the traffic injected from the main links is bounded and negligible. For this purpose we propose an ultra-low latency logarithmic interconnect designed following AXI-4.0 standard and present an analysis on the role of this interconnect in SMC. Next, our CA model for the baseline HMC system and its SMC extension are presented. As shown in Figure 4.1, the main interconnect and the vault controllers are two key components in design of the smart memory cube. In this section, we present the design and calibration methodology for these two key components.

4.3.1 The Main Interconnect on LoB

We have designed our interconnect based on the ultra low-latency "logarithmic interconnect" (originally designed for L1/L2 contexts in chapter 2 and chapter 3), and modified it to support high bandwidth communication. Also, AMBA AXI 4.0 standard [96] has been chosen, which is the most widely used standard for on-chip communication in system-on-chips. This standard divides traffic streams into 5 categories and dedicates independent channels to each of them (AR: Address Read, R: Read Data, AW: Address Write, W: Write Data, B: Write Response).

Figure 4.2 illustrates a high-level schematic view of our interconnect design. When a transaction arrives at the AXI master ports, first the "Issue Logic" decides whether it should be allowed to enter the memory system. The Issue Logic limits the maximum outstanding transactions of each master port to a certain number called MoT, and assigns unique tags to the ones allowed into the interconnect. It will be shown in section 4.5 that MoT has an important effect on performance and avoiding the system from going into saturation. Next, address remapping/scrambling is performed on the transactions based on the intended addressing scheme. WRITE and READ transactions arrive through independent AXI channels. AW and W channels deliver address and data for the WRITE transactions, and after address decoding and identifying the destination port (inside the "Master Blocks"), WRITE address will be sent to an arbi-

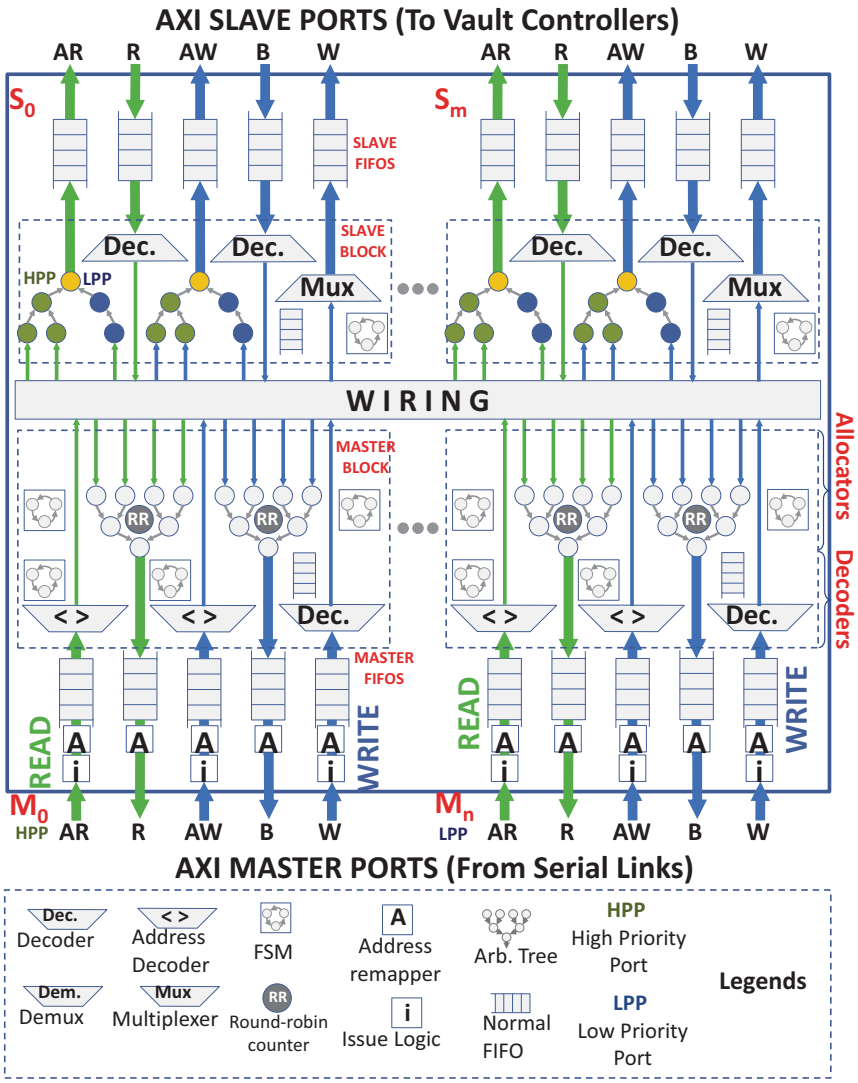


Figure 4.2: Proposed AXI 4.0 based logarithmic interconnect for SMC

tration tree. Among the master ports and separately on the PIM ports fair round-robin arbitration is performed. However, in the last stage of the network a fixed-priority arbitration scheme ensures higher priority for the main ports (hierarchical arbitration). This way only residual bandwidth is delivered to the PIM. The winner request will have its data delivered through multiplexers in the "Slave Block" to the FIFOs on the slave ports. Since AW and W channels do not have to be synchronized in AXI-4.0 standard, a transaction FIFO inside the "Slave Block" associates pending addresses to data bursts. A similar procedure takes place for READ transactions, except that READ requests do not have any data associated with them, and they are 1 flit long. On the other side of the network (slave side), B and R channels deliver back the response data, and acknowledgement of the WRITE transaction, respectively. The responses

will arrive at the intended "Master Block" after a simple decoding, and there, they will wait for arbitration with other responses destined to the same master. Finally, the responses arrive at the intended master port via R and B channels. All arbitrations performed inside this network are single-cycle, and the whole interconnect has been developed as a synthesizable and parametric RTL model.

4.3.2 Address Remapping and Scrambling

As the memory space of an HMC device is split across multiple independent vaults, an address mapping scheme is required that determines which address is mapped to which vault. HMC supports different types of address mapping through its Address Mapping Mode Register. The default address mapping in HMC is low-interleaved [13]. In our model, address interleaving scheme can be modified through "Address Remapper" modules illustrated in Figure 4.2. It is also possible to scramble the incoming addresses. The baseline address mapping of HMC is [RC.BA.VA.OF] (VA: vault address, BA: bank address inside the vault, RC: row and column addresses, and OF: offset bits of the address). Assuming that transaction splitting is not possible in the HMC [13], a full transaction is always directed to one bank, therefore, OF bits are always in the least significant position. This results in 6 possible permutations for conventional address mapping.

As the internal organization of an HMC can be hidden to its users, address mapping schemes need not be limited to the conventional schemes. The choice of the address mapping scheme matters because the parallelism offered by the HMC can only be fully exploited if the memory requests are evenly spread across the vaults. Unfavorable memory access patterns may lead to performance limitations. The primary goal of an address mapping scheme is thus to avoid vault/bank conflicts. For sequential memory accesses, interleaving performs very well because it spreads the requests evenly across the vaults and therefore maximizes parallelism. But for other memory access patterns, interleaving may not optimal because it may fail to distribute the requests evenly across the vaults. While it is possible to come up with address mapping schemes that are tuned to specific memory access patterns [131][128], address scrambling follows a different idea: it tries to turn most memory access patterns into random-looking access patterns [132]. This is motivated by the observation that while memory access patterns may vary over time, it is difficult to change the address mapping scheme of an HMC at runtime. Before actual implementation and cycle accurate simulation, we evaluate our proposed address scrambling mechanism in terms of randomness. Then in

subsection 4.5.2 we use cycle-accurate simulation for performance analysis and discuss about unfavorable patterns which cause troubles to the default address mapping of HMC.

Construction Principle

We use discrete Fourier transform (DFT) as a heuristic for construction of our address scramblers. This is motivated by three reasons: First, the DFT can be computed efficiently, for example by the FFTW library [133]. Second, it is an invertible transform, which means that it does not lose any information. Third, the DFT of a random vector is expected to be flat except possibly for the DC component. In fact, calculation of DFT is one of Spectral Density Estimation (SDE) methods for the stochastic signals [134]. If we define the DFT of a sequence of N numbers as

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i k n}{N}} \tag{4.1}$$

and let the x_n be independent and identically distributed. Then, the expected squared magnitude of X_k does not depend on k as long as k is nonzero:

$$\begin{aligned}
\mathrm{E}\big[|X_k|^2\big] =& \mathrm{E}\big[X_k \overline{X_k}\big] && (4.2)\\
=& \mathrm{E}\left[\frac{1}{N} \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} x_n x_{n'} e^{-\frac{2\pi i k (n-n')}{N}}\right] && (4.3)\\
=& \frac{1}{N} \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} \mathrm{E}[x_n x_{n'}]\, e^{-\frac{2\pi i k (n-n')}{N}} && (4.4)\\
=& \frac{1}{N} \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} \left(\mathrm{Cov}(x_n, x_{n'}) + \mathrm{E}[x_n]\mathrm{E}[x_{n'}]\right) e^{-\frac{2\pi i k (n-n')}{N}} && (4.5)\\
=& \frac{1}{N} \sum_{n=0}^{N-1} \mathrm{Var}(x_n) + \frac{1}{N} \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} \mathrm{E}[x_n]\mathrm{E}[x_{n'}] e^{-\frac{2\pi i k (n-n')}{N}} && (4.6)\\
=& \sigma^2 + \frac{\mu^2}{N} \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} e^{-\frac{2\pi i k (n-n')}{N}} && (4.7)\\
=& \begin{cases} \sigma^2 + N\mu^2 & \text{if } k = 0, \\ \sigma^2 & \text{if } k \neq 0. \end{cases} && (4.8)
\end{aligned}$$

Here, (4.4) follows from the linearity of the expectation, (4.5) follows from a property of the covariance, (4.6) follows because the x_n are independent, (4.7) follows

because the x_n are identically distributed with mean μ and variance σ^2, and (4.8) follows from a simple calculation. The address scramblers are assessed in the following way: Let x_n be the sequence of all scrambled addresses, i.e., let $x_n = \phi(n)$ for all $n \in \{0, \ldots, N-1\}$ where ϕ denotes the address scrambler and where n is interpreted as an address. Address scramblers with a smaller peak magnitude

$$\max_{k \in \{1,\ldots,N-1\}} |X_k| \tag{4.9}$$

are considered to be better (the DC component X_0 is intentionally left out). This is motivated by the observation that the DFT tends to be flat if (4.9) is small. Note that the flatness criterion itself is only a heuristic because although (4.8) makes a statement about every single X_k, it does not say anything about their joint distribution, also the statistical independence of the input samples assumed in (4.6) is not always true.

Construction Methodology

An address scrambling function is a permutation on the set of all addresses. One way to obtain it is to use a substitution-permutation network as depicted in Figure 4.3. Substitution-permutation networks are widely used in the design of cryptographic block ciphers [135]. They are inherently parallel and can achieve good diffusion properties. The substitution-permutation networks in our design use the 4-bit S-box from the PRESENT cipher [136], which is optimized for hardware efficiency. Since the cryptographic properties are not required and because the block size of PRESENT is larger than the number of address bits, custom substitution-permutation networks are used.

The custom substitution-permutation networks are constructed layer by layer as follows: Initially, the network is empty and a set of candidate permutations on the address bits is generated at random. For every candidate permutation, a candidate network is built consisting of the existing network, the candidate permutation, and a layer of S-boxes. An S-box is an invertible mapping between a 4-bit input and a 4-bit output [136]. The candidate networks are assessed according to the criterion based on the discrete Fourier transform which was described earlier. The candidate network with the smallest peak magnitude is selected, and the procedure is repeated until the network has the desired number of layers. Finally, the address bits that are mapped to the vault bits are determined: all combinations are evaluated and the combination with the smallest peak magnitude is selected. This process is used to construct an initial address scrambler for 22 address bits and 5 vault bits. For different number of layers,

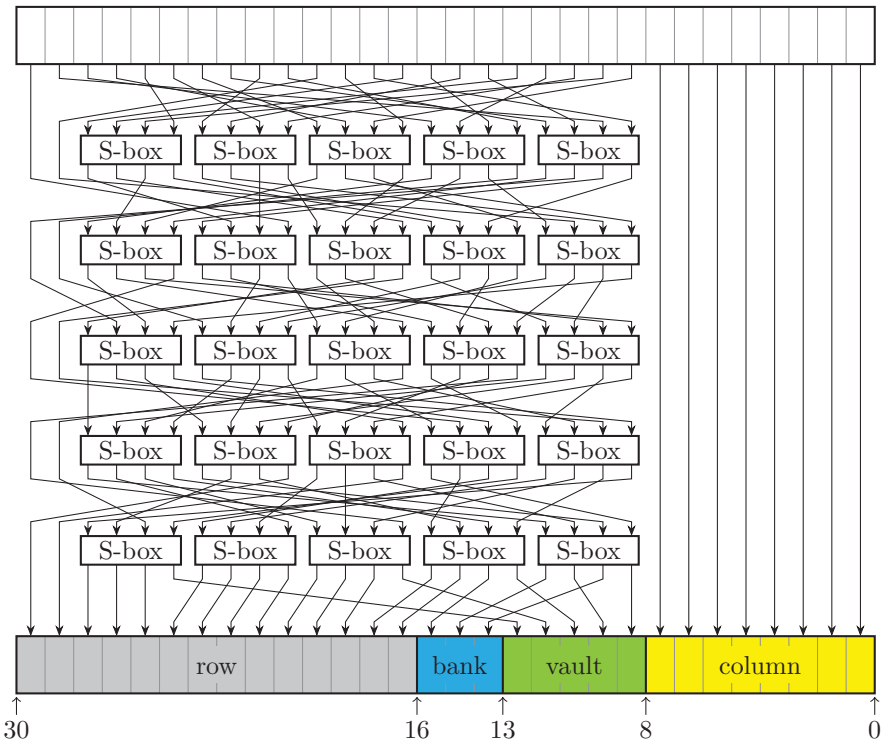


Figure 4.3: 5-layer substitution-permutation network for 22 address bits (S.22.5.05)

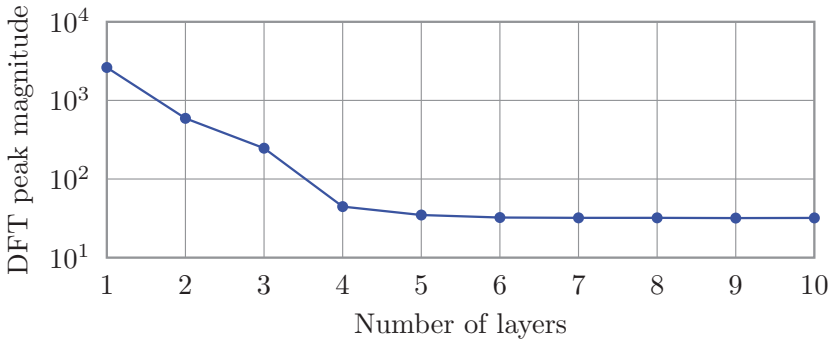


Figure 4.4: Quality of 22-bit substitution-permutation networks vs. number of layers.

the DFT peak magnitude for the address-to-vault mapping is shown in Figure 4.4. For the performance evaluation, the 5-layer network depicted in Figure 4.3 is selected.

To improve the sequential read performance, the network is modified as follows. The vault bits do not take part directly in the address scrambling. Instead, they are left out and XORed together with some bits from the output of the address scrambler as illustrated in Figure 4.5. The substitution-permutation network is constructed similar to previous case. This construction guarantees that the data of an 8 KB block is evenly

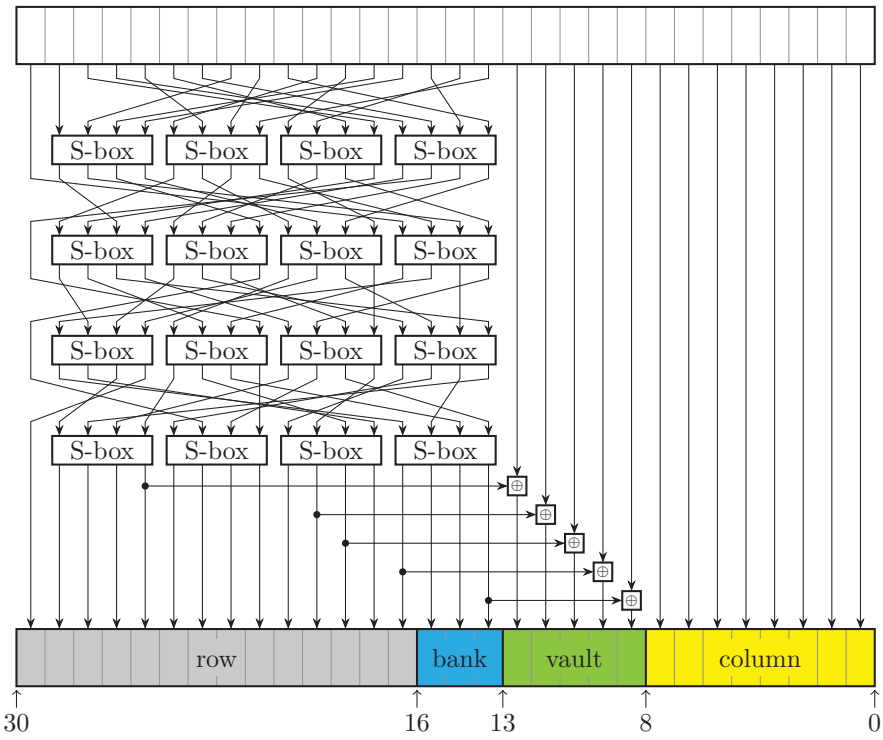


Figure 4.5: Modified 4-layer address scrambler for 22 address bits (S.17.5.04)

distributed across the vaults.

Further improvements are possible. The construction depicted in Figure 4.6 is a valid address scrambler because it implements a bijective function. It has the same address-to-vault mapping as the address scrambler shown in Figure 4.5. Since the performance is expected to be dominated by the vault conflict behavior and because both address scramblers have the same address-to-vault mapping, they should have roughly the same performance (See subsection 4.5.2 for results). The construction from Figure 4.6 has the advantage of not modifying the MSBs of an address. Address scrambling can therefore be enabled or disabled independently for every 8 KB block, which is illustrated in the figure. It is for example possible to have an additional bit in the incoming transactions that specifies whether scrambling is enabled or not for it. Operating system or hardware must ensure that all users of an 8 KB memory block agree whether scrambling is enabled or not.

We would like to mention here that, a mathematical proof for bijectiveness of the proposed scrambling schemes is out of the scope of this thesis, nevertheless, we have verified the bijectiveness of all proposed scrambling schemes exhaustively and all mappings were found to be one-to-one. In this chapter, we evaluate 6 address scrambling

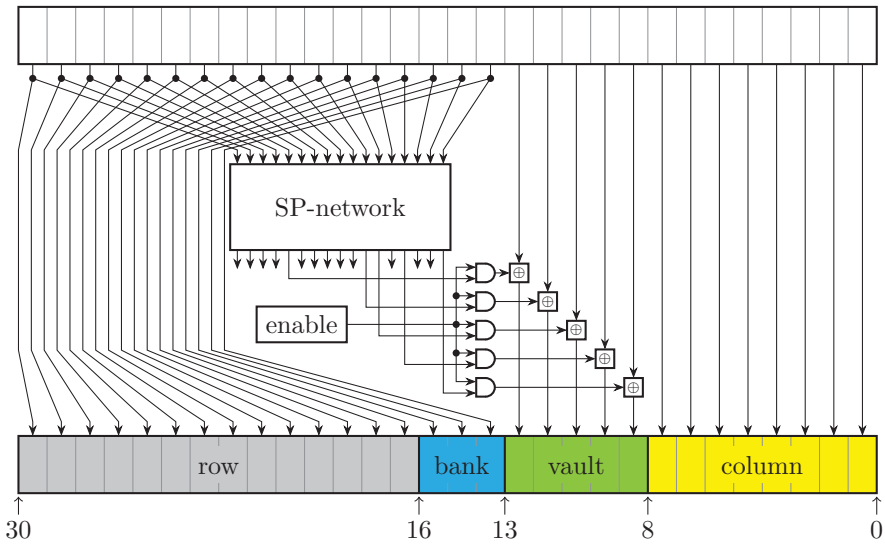


Figure 4.6: Address scrambler for improving sequential read with additional signal to enable or disable scrambling (S.17.5.04.T)

alternatives: *S22.5.05* is the original design shown in Figure 4.3, with 22 scrambling bits, 5 vault bits, and 5 layers. *S.17.5.04* is illustrated in Figure 4.5, and *S.17.5.04.T* is shown in Figure 4.6, both having 4 scrambling layers. We have created three alternative schemes, as well. *S.14.8.04* uses 14 bits (RC) for scrambling and XORs 8 bits with them (BA + VA bits) (Figure 4.7.a). The idea is to improve bank-level parallelism in presence of linear traffic patterns. *S.14.8.04.T* is designed based on *S.14.8.04* with the RC bits bypassed (See Figure 4.7.b). This allows for enabling/disabling the scrambler for every 64 KB block, while its performance is the same as *S14.8.04*. Finally *S.14.8.00* has zero scrambling layers. It only XORs random RC bits with BA and VA bits (Figure 4.7.c). The obtained results are presented in section 4.5.

4.3.3 Vault Controllers

Design of the Vault Controllers follows a bank-parallel DRAM Channel Controller, again with a standard AXI 4.0 interface to connect seamlessly to the main interconnect (See Figure 4.8). The first stage in this channel controller is a round-robin arbitration among the AXI AW and AR channels to issue one of them to the Command Queue (CMDQ) in every cycle. In case of WRITE, the burst data is stored in the WData FIFO. A set of Finite State Machines (FSMs) control power up/down, auto-refresh, and configuration of the DRAM devices; and for each memory bank a Read-Write FSM keeps track of the bank state and timings such as t_{RAS}, t_{RCD}, t_{RP}, t_{WR} using a set

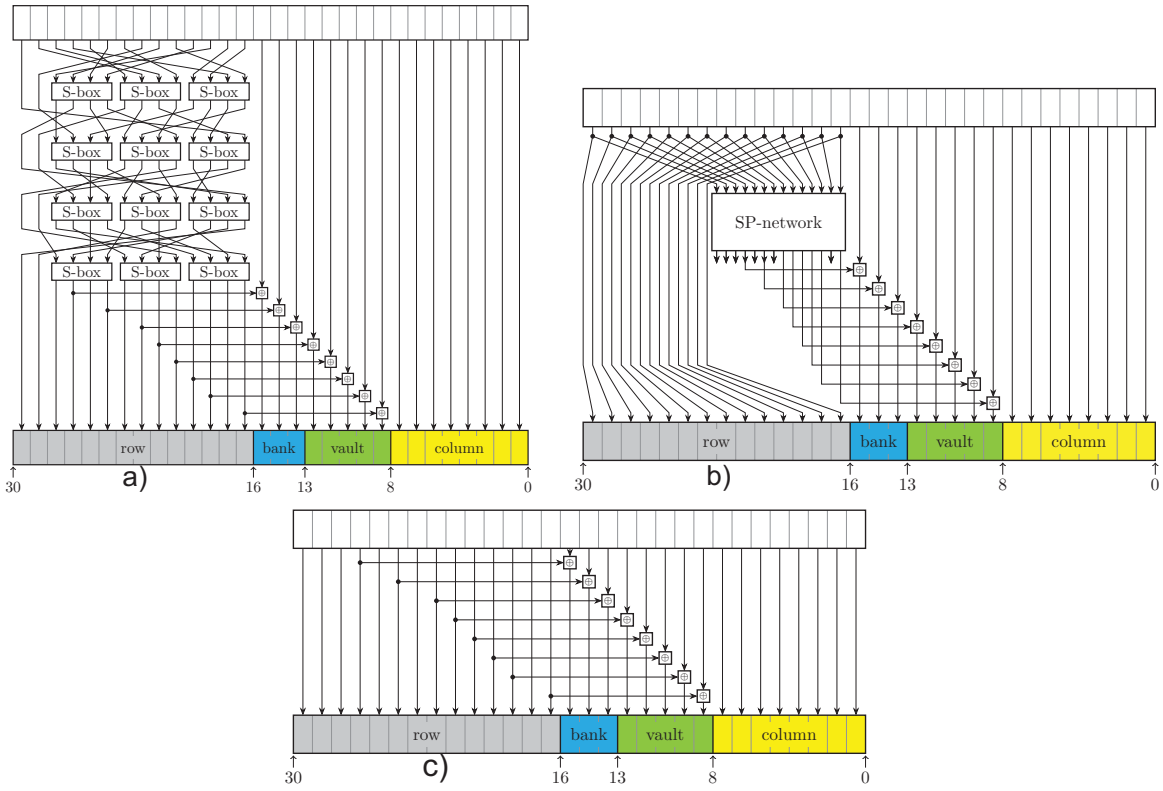


Figure 4.7: Three additional scrambling alternatives: *S.14.8.04* (a), *S.14.8.04.T* (b), *S.14.8.00* (c).

of counters. Bank level parallelism is implemented by the aid of these independent FSMs and transaction queues. Finally, one Master FSM controls the main DRAM bus and all other mentioned FSMs. Design of the vault controllers and the signalling at the DRAM bus follows the JESD79F JEDEC standard for DDR SDRAMs [137], in a generic and configurable way. Moreover, different techniques of pipelining and latency hiding have been implemented to reach the highest throughput; and unlike the standard vault controllers in HMC, this model supports both open and closed page policies. AXI Interface returns the READ response and WRITE acknowledge through R and B channels respectively. B channel simply acknowledges receive of full WRITE burst. And for READ transactions, after receiving a complete response from the DRAM device, converts it to AXI Burst in the and returned through the AXI R channel. Since data widths and burst sizes of the AXI interconnect and the DRAM devices do not necessarily match, this module performs burst size conversion, as well. In addition, the clock domains of the AXI interconnect and the vault controllers have been designed independently, therefore, Command Queues (CMDQ) and RData FIFOs are asynchronous dual-clock FIFOs [138] to ensure data integrity.

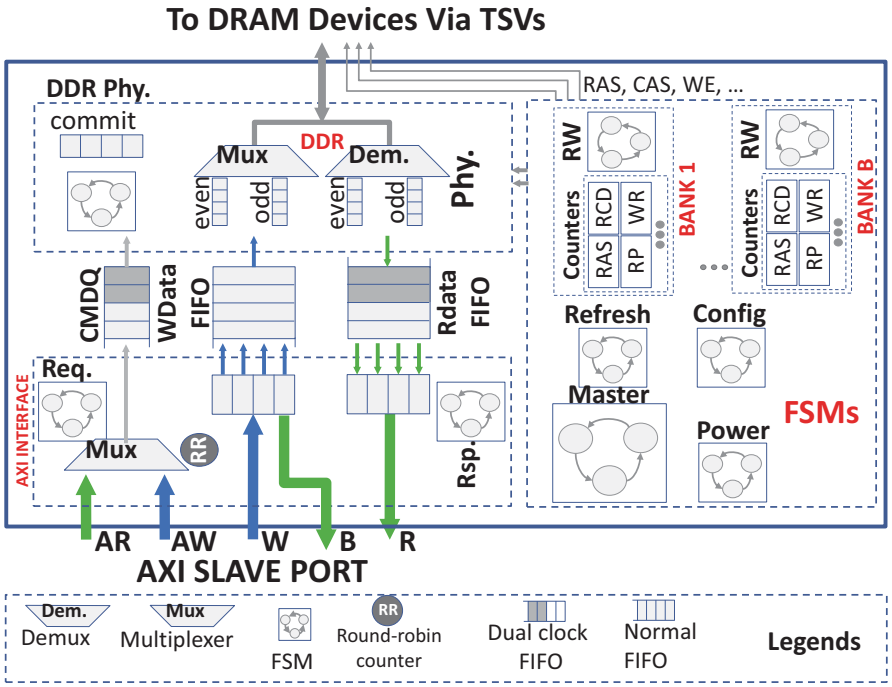


Figure 4.8: Schematic view of a Vault Controller

The DRAM device models have been adopted from [139] and extended in terms of number of banks, burst size, data width, etc. Detailed design of the serial-link controllers is extensively studied in [140]. In this chapter, we focus on traffic management inside the memory cube, therefore we won't model the serial links. However, in chapter 5, a higher level serial link model will be developed and utilized. One final point is that, standard and flexible design of our main interconnect allows for connecting any AXI-compatible device including parallel processor-in-memory clusters. Therefore, PIM can be easily integrated in this model by simply increasing the number of low-priority master ports of the main interconnect and attaching the PIM device to them (See Figure 4.1).

4.4 Calibrating The CA Model

In the latest specifications [13] for a prototype of HMC, 4 to 8 Memory dies and 1 Logic Base (LoB) have been reported, with 32 independent memory vaults each consisting of 2 DRAM banks per memory die. With a bank size of 4MB a total memory of 1GB/2GB is provided in this configuration. Each vault is expected to deliver a bandwidth of 10GB/s to the lower LoB. This aggregates in total to a maximum of 320GB/s. On the other side, four serial links consisting of 16+16 differential lanes for READ and

WRITE are provisioned [140]. With a lane bit-rate ranging from 12.5 Gbps to 30 Gbps an aggregate off-chip bandwidth ranging from 200 GBytes/sec to 480 GBytes/sec. can be delivered. Furthermore, in the HMC Specifications [13] it is implied that the flit size of the link layer is equal to 16 Bytes. Given that the largest packet serialized on an HMC link contains 16 data flits and 1 control flit (8 Bytes header + 8 Bytes tail), the packetization overhead over the serial links can be estimated as 6.2%. We should mention that HMC links advocate a data scrambling mechanism to avoid a run of many consecutive zeros (or ones) for sake of clock recovery at the destination. If the scrambled data on any lane exceeds 85 similar digits, the transaction is corrupted intentionally and a link retry is requested. The probability of such event is approximated by 2^{-80}. Given these details, and that maximum down bandwidth from the vaults is 320 GB/s, HMC serial links should deliver up to 300GB/s of useful data (read+write). It is worth mentioning here that the flits are transferred over the serial lanes in a bit-interleaved fashion [13]. Therefore in each Unit Interval (UI) of data transfer, 16 bits of the same flit are transferred to the destination and a whole flit would require 8 UIs. This can provide a flexibility for size conversion before the interconnect, which will be studied in section 4.5.

In the first paper on HMC [91] 32 data TSVs were reported per each vault with a double data rate (DDR) data transfer mechanism, this requires a clock frequency of 1.25GHz to deliver 10GB/s [54]. Also, unlike existing DDR memories, HMC utilizes Closed-Page policy and its DRAM devices have been redesigned to have shorter rows (256 Bytes matching the maximum burst size of serial links, rather than 8-16KB in a typical DDR3 device) [91]. This is because HMC has been mainly designed for High Performance Computing (HPC) and server workloads which typically exhibit little or no locality, either due to the underlying algorithm (e.g., pointer chasing or sparse floating point computations) or the execution model (e.g., highly threaded server workloads). The reduced row length helps save power by alleviating the over-fetch problem, however, reduces the row buffer hit probability, which makes open page mode impractical. In addition, open page policy exhibits additional overheads for little locality workloads, due to delaying the precharge between accesses to different rows [54][121] (See subsection 4.5.1 for experiments). Open page row buffer models also impose a logic cost as the scheduling hardware is typically more complex [54]. As a result, with the large number of banks in HMC, it is more efficient to utilize memory-level parallelism to achieve high performance rather than relying on locality which may or may not be present in a given memory access stream. This also motivates our address scrambling

proposal to increase the randomness of the incoming address patterns.

The specifications of the DRAM devices utilized in HMC are proprietary. However, to the best of our knowledge [141] contains the most comprehensive set of parameters that currently published for HMC: {t_{RP}=13.75ns, t_{tCCD}=5ns, t_{RCD}=13.75ns, t_{CL}=13.75ns, t_{WR}=15ns, t_{RAS}=27.5ns}. Moreover, we assume that the DRAM devices have the same clock frequency ($t_{CK} = 0.8ns$) as the TSVs, while the interconnect and the rest of the cube work in different clock domains. Besides, we assume the internal Data Width of the DRAM devices to be 32 bits, matching the TSV count. These assumptions simplify the 3D interface allowing for direct connection between TSVs and the DRAM devices. Similar approach has been taken in [54].

The main interconnect in HMC should connect 4 serial links to 32 memory vaults. This asymmetry makes the cardinality of this interconnect non-trivial. For example with a typical clock frequency of 1GHz and a flit size of 128-bits, a total of 128GB/s bandwidth can be delivered with a cardinality of 4x32 (which is below the intended limit). To cope with this problem, we increase in the number of master ports in the main interconnect from 4 to 8, connecting each Link Controller to two of them. Moreover, we increase the flit-size of the main interconnect from 128 to 256 bits. This way can achieve a theoretical aggregate bandwidth of 512GB/s at the master side (which is beyond the requirements of current and future HMC systems). Therefore, each serial link breaks the incoming transactions into two chunks and forwards them via 2 AXI Master ports. Size conversion from 128-bit to 256-bits flits is also done easily in the Link Controllers where serial data is being converted to parallel. The only issues which should be considered are the clock frequency of the interconnect and the increase in its area due to increased flit-size. Throughout this chapter the interconnect has a cardinality of 8x32 with a flit-size of 256, unless otherwise stated. With this assumption, the maximum burst size inside the interconnect will be reduced to 8. Next section presents the experimental results.

4.5 Experimental Results

Our baseline HMC model has been described using SystemVerilog HDL in a cycle-accurate and fully synthesizable style. ModelSim has been utilized for simulation, and logic synthesis has been performed using Synopsys Design Compiler using STMicroelectronics Bulk CMOS-28nm Low Power technology library. Area of each vault controller was found to be $0.62mm^2$, and the AXI interconnect less than $0.4mm^2$. Summing to

a total of about $20.3mm^2$ which is even less than the DRAM area reported for HMC ($68mm^2$) [91] for 16 vaults. For 32 vault, DRAM area will be even larger, so there will be even more space for near memory computation. Power consumed in the interconnect was found to be less than 5 mW up with a clock frequency up to 1 GHz.

Three groups of memory access traces are used to characterize the designed system: Synthetic traffic generated in ModelSim (random, linear, localized to banks/vaults), Sparse Graph Traversal computation kernels (Average Teenage Follower, Bellman-Ford Shortest Path, and Page-Rank [122]), and PARSEC V2.1 benchmark suite [101]. Dense matrix addition is utilized as well, for its high bandwidth demand and low memory access time sensitivity. The non-synthetic traces are gathered in *gem5* [100] running a full-system simulation of eight x86 CPUs with Linux 2.6.22.9 kernel, at the last level cache (LLC) port. As previously observed in [54], we noticed that current benchmarks cannot easily utilize the full bandwidth provided by HMC. To increase the bandwidth demand of the traces, we used trace time compression. This is representative of future systems with much more bandwidth requirements (a common trend).

4.5.1 HMC Exploration

First we adjusted the size of all buffers along with the MoT parameters to achieve the maximum bandwidth requirement of HMC with a reasonable access latency, measured as Average Memory Access Time (AMAT). Through several experiments we realized that only the size of the CMDQ FIFOs in the vault controllers (Figure 4.8) and MoT (Figure 4.2) affect the delivered bandwidth and latency significantly, while the size of the other FIFOs can be minimized. We found that with CMDQ Size=32 elements, and MoT=44 up to 205 GB/s (READ) can be achieved for maximum pressure uniform random traffic and up to 255 GB/s for synthetic linear traffic. With these numbers, Figure 4.9.a,b illustrate delivered bandwidth and AMAT versus requested bandwidth, in one vault and in the baseline HMC model, respectively. Uniform random read transactions have been applied to the ports, and AMAT has been measured at the response master ports of the interconnect. As this figure shows, when the network is not saturated, delivered bandwidth is over 99% of the requested bandwidth (199GB/s), beyond HMC's intended read bandwidth (160GB/s). While AMAT is bounded and less than 300 ns, which is about 4X of the zero load read latency (76ns) illustrated in Figure 4.9.c. Note that 83% of the zero load latency is related to internal DRAM timings, also since a write transaction is immediately acknowledged without waiting for completion, zero-load latency of a write transaction does not include DRAM timings

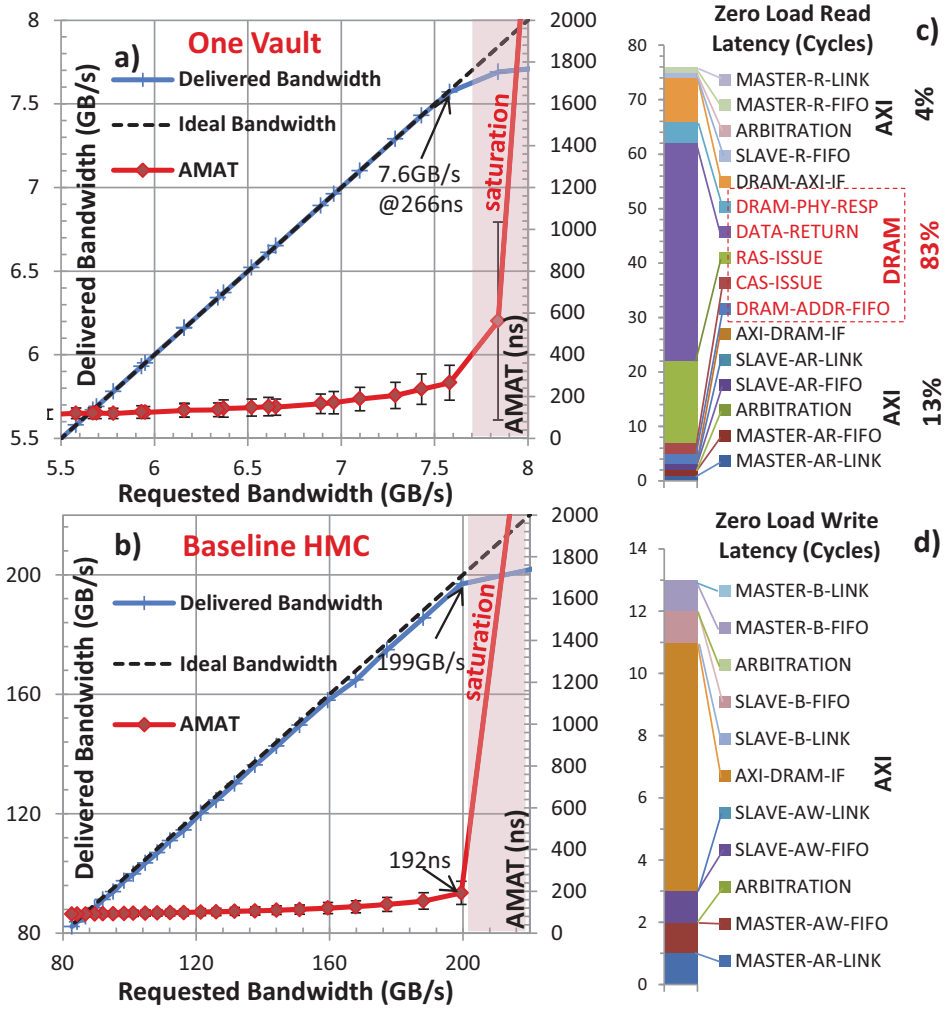


Figure 4.9: Delivered bandwidth of one vault only (a) and the baseline HMC (b). Zero load latency breakdown for READ (c) and WRITE (d) commands.

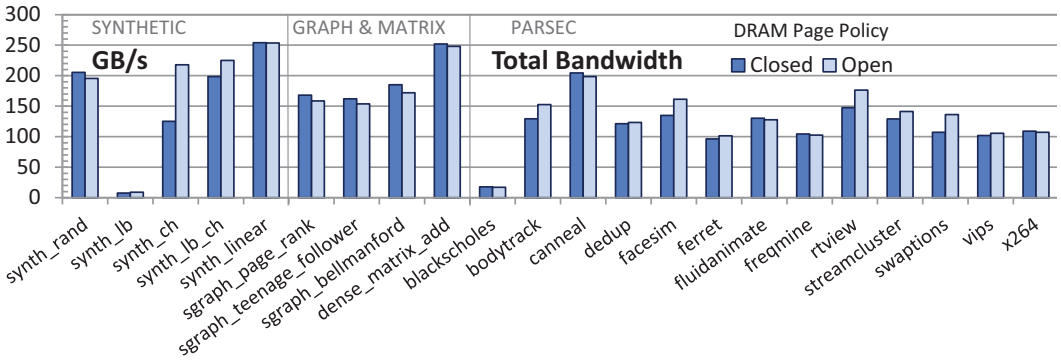


Figure 4.10: Effect of page policy on delivered bandwidth from SMC (Figure 4.9.d).

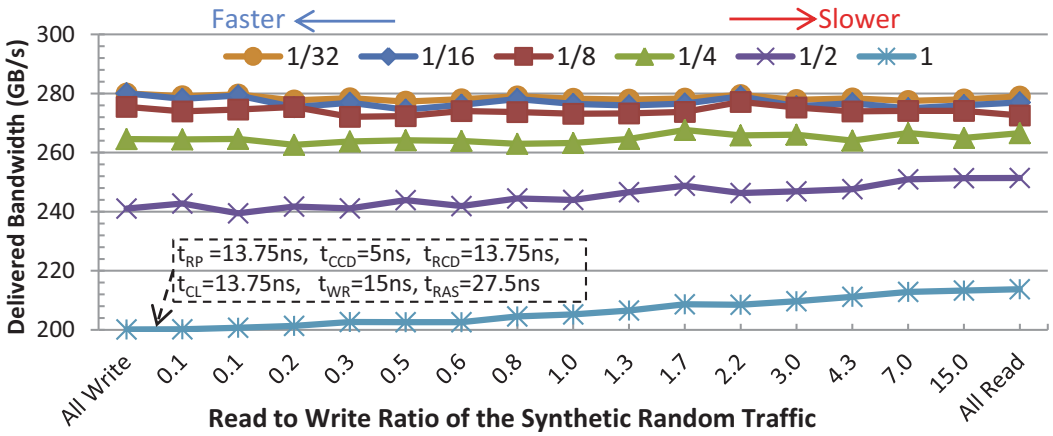


Figure 4.11: Effect of R/W ratio of random traffic on total bandwidth delivered from SMC

To study the effect of page policy on delivered bandwidth, different traffic patterns have been applied to the baseline model changing only the page policy, with results plotted in Figure 4.10.a. *synth_rand* is a uniform random read traffic with only read transactions of maximum burst size. *synth_linear* is a synthetic linear read traffic pattern with steps of 256 Bytes. *synth_lb*, *synth_ch*, and *synth_lb_ch* are special read random traffic patterns with address bits set to zero except for bank bits, vault bits, and bank+vault bits, respectively. These traffics are designed to exercise banks/vaults/banks+vaults respectively. It can be seen that *synth_linear* can achieve a read bandwidth of 255 GB/s regardless of the page policy. This is the maximum bandwidth which can be extracted from the system, and as explained later, it is limited by the row-cycle time of the DRAM devices. Interestingly, when *synth_rand* traffic is applied, closed-page policy operates better than open-page, with a delivered bandwidth beyond requirement of HMC (i.e. 205GB/s for READ). And even in PARSEC benchmarks which have a lot of locality, still open page does not provide superior benefits to cover for its implementation cost.

Next, synthetic random traffic has been injected and read to write ratio has been changed to study the effect of mixing reads and writes. Results are shown in Figure 4.11. In this plot, DRAM timings have been scaled down artificially from 1X down to 1/32X. This is to show how much performance is lost due to the internal timings of the DRAM devices. For 1/32 scale value DRAM latencies become negligible. It can be seen that injecting mixed read and write traffics does not improve the delivered bandwidth, in comparison to "All Read" and "All Write" traffics. This is due to the serialization bottleneck of the read and write transactions in the DRAM Bus

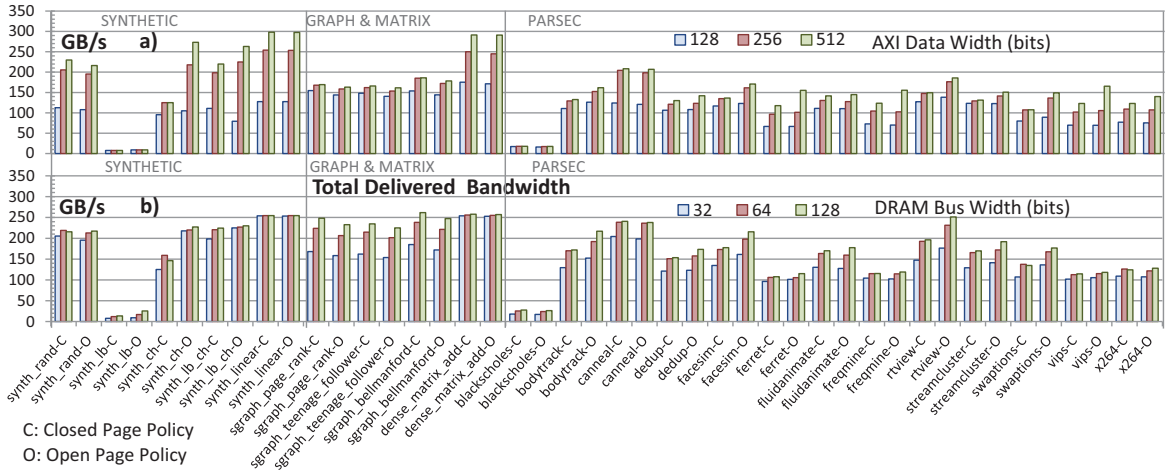


Figure 4.12: Effect of AXI data width (a) and DRAM bus width (b) on delivered bandwidth from SMC

and FSMs. Also, looking at two corners of the plot reveals that "All Read" traffic achieves a slightly higher bandwidth than "All Write" (when DRAM timings are not scaled down). This is because the minimum distance between two subsequent read transactions is limited by the *row-cycle-time* which is $(t_{RAS} + t_{RP})$, while for write transactions *write-recovery time* (t_{WR}) should be satisfied, as well.

Next, we analyze the effect of different architectural parameters on the delivered bandwidth. This is to find the minimum parameters meeting the performance requirements of current and future HMC releases. Figure 4.12.a illustrates delivered bandwidth when the flit size of the AXI interconnect is changed from 128 to 512. We can see that, 128-bit flits are not enough (specially for the random traffics), while, 512-bits is not necessary for closed-page policy. Moving from 256 to 512 flits results in 7% average performance improvement for PARSEC and less 1% for the graph benchmarks. Note that graph benchmarks exhibit almost random traffic patterns with a very low locality. Figure 4.12.b illustrates the effect of DRAM Bus Width (i.e. number of TSVs per each vault). For closed-policy, changing the TSVs from 32 to 64 results in an average performance improvement of 23% for PARSEC and 31% for the graph benchmarks. This suggests that designing smaller TSVs to have wider links between DRAM dies is an interesting direction in evolution of 3D integration, and can result in further performance improvements.

Next, the clock period of the DRAM devices (t_{CK}) and the AXI interconnect have been swept and Figure 4.13.a,b illustrate their effect on delivered bandwidth. The goal is to find their optimal clock frequency. Increasing DRAM clock frequency from

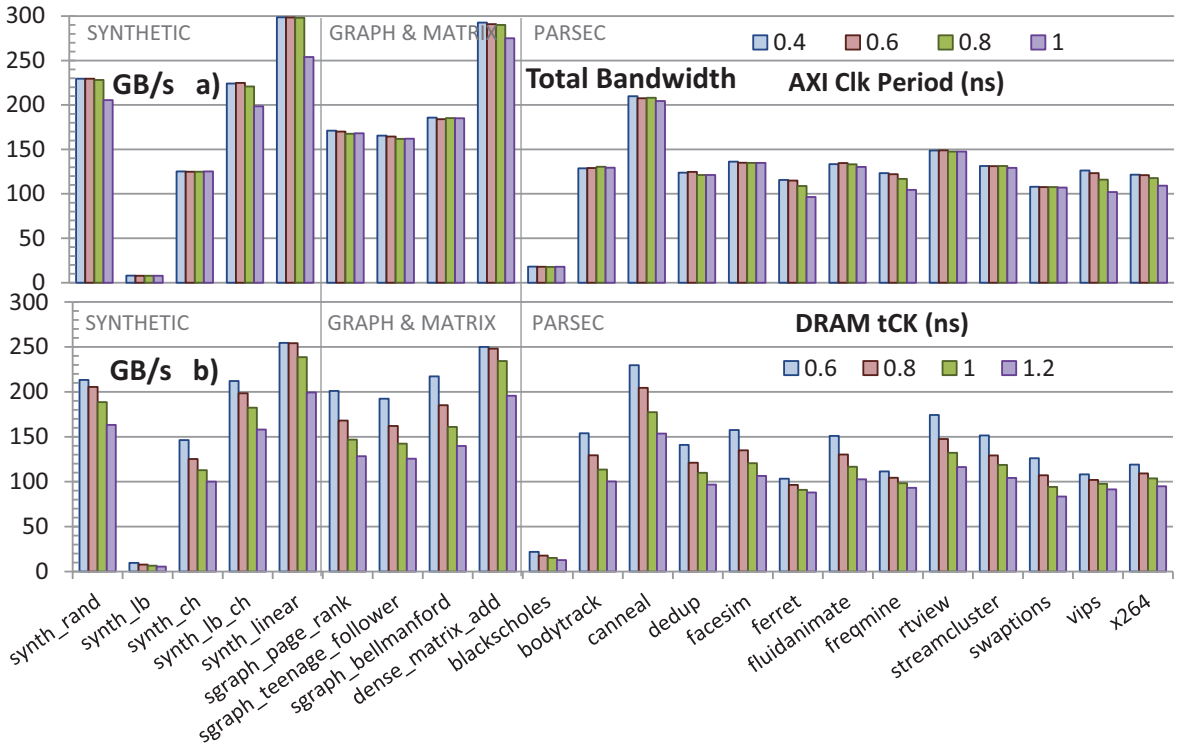


Figure 4.13: Effect of DRAM t_{CK} (a) and AXI Clk period (b) on bandwidth delivered by SMC

830 MHz to 1.6 GHz can improve delivered bandwidth by over 40% for PARSEC, 50% for graph benchmarks, and 30% for synthetic random traffic, while the clock period of AXI Interconnect does not have much effect (less than 2% performance improvement by increasing interconnect's clk frequency from 1GHz to 2.5GHz). Again this shows that the main bottleneck of the system is the DRAM and not the AXI interconnect. To further confirm this result we have scaled down all the timing parameters of the DRAM devices from their default values by the scale factors illustrated in Figure 4.14.a, where the same traffic set has been applied. Interestingly, the delivered bandwidth to PARSEC and graph benchmarks can be highly improved when DRAM timings are very small and close to an ideal SRAM (37% and 52% improvement when timings are scaled by 16). This is while, random patterns are less sensitive to DRAM timings (less than 15% on average).

Figure 4.14.b depicts the effect of the number of banks per each vault on the delivered bandwidth. When there is only one bank per each vault, closed-page behaves almost similarly to the open-page, while as this number increases, closed-page performs slightly better. Overall, an average performance improvement of 15% for PARSEC and 58% for the graph benchmarks is achieved for closed-page by increasing the number of

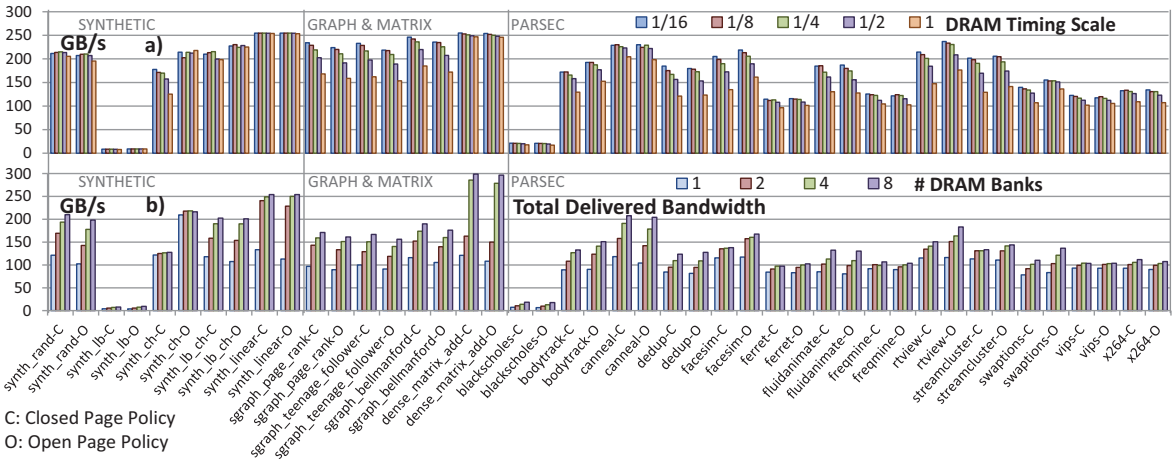


Figure 4.14: Effect of scaling down the timing parameters of DRAM (a) and number of banks per vault (b) on bandwidth delivered by the SMC

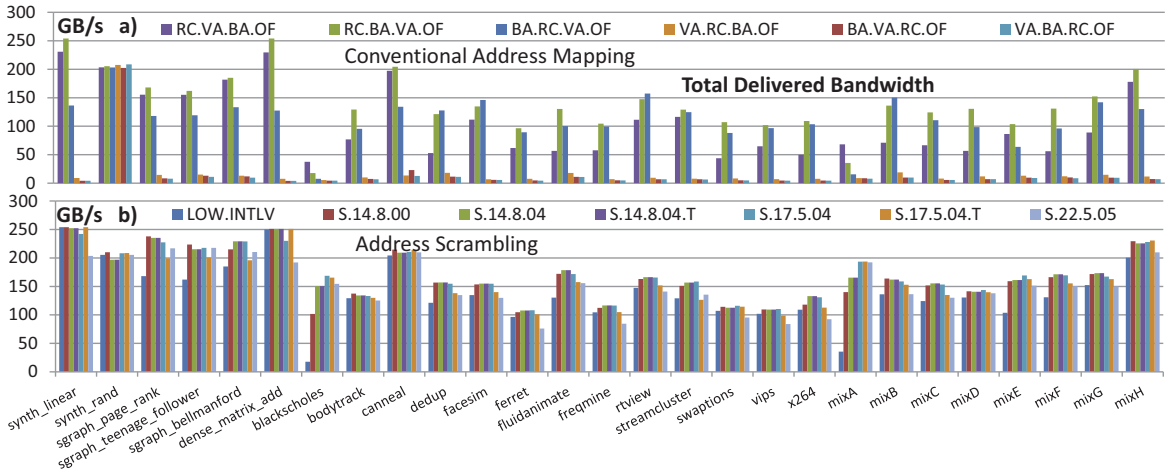


Figure 4.15: Effect of conventional address remapping (a), and address scrambling (b) on delivered bandwidth of SMC

banks in each vault from 1 to 8. This can be explained using the fact that graph benchmarks exhibit more random access patterns, therefore can benefit more from bank-level parallelism.

4.5.2 Address Remapping

For traffics which exhibit locality, address mapping was found to have an extremely important effect on the delivered bandwidth and total execution time. Figure 4.15.a illustrates the effect of conventional address mapping on total delivered bandwidth to

different benchmarks. Benchmarks starting with *mix* prefix are a mix of different graph and PARSEC benchmarks simulating a multi-programmed workload. Interestingly, address mapping scheme affects the benchmarks differently. For x264 the best address mapping is RC.BA.VA.OF (HMC's default mapping [13], explained in subsection 4.3.2), while in rtview VA.BA.RC.OF achieves the highest performance. In dense matrix addition RC.VA.BA.OF achieve the highest bandwidth.

In the next experiment, conventional address remappers have been replaced with address scramblers. Results are plotted in Figure 4.15.b. *LOW-INTLV* is the default address mapping of HMC (i.e. RC.BA.VA.OF), and all scrambling methods have been described in subsection 4.3.2. On average for PARSEC, address scrambling alternatives improve bandwidth over the *LOW-INTLV* by a factor of 88% (note that most of this improvement is related to blackscholes, which we will investigate later). For graph benchmarks this improvement is around 24%. Also, looking at the mixed benchmarks, reveals that multiprogramming slightly reduces the benefits achieved by scrambling (84% average bandwidth improvement). This shows that mixing different traffics slightly improves their randomness. Lastly, all proposed scrambling mechanism perform better than the baseline *LOW-INTLV* on average. Original S.22.5.05 results in an average performance improvement of 7%, while the two improved versions S.17.5.04 and S.17.5.04.T lead to 20% and 16% improvement on average over all benchmarks. One last point to mention is that even though S.14.8.00 delivers a reasonable bandwidth in Figure 4.15.b, it will be shown later in Figure 4.18 that it is vulnerable to adverse traffic patterns, and therefore, it is not a suitable candidate.

In the next set of experiments, we aim to analyze the traffic patterns which are specifically unfavorable for the *LOW-INTLV* addressing and present examples of real applications generating these patterns. In Figure 4.16, a synthetic linear traffic pattern has been applied to the baseline HMC model (with *LOW-INTLV* addressing scheme) and the step size of the traffic has been changed from 1 DRAM row (256 Bytes) to 2048 DRAM rows (512 KBytes). It can be seen that increasing the step size drops the delivered bandwidth severely and increases total execution time. The reason is that by increasing the step from 256 to 512 we are skipping half of the memory vaults and using only the even indexed ones. This happens also for further increase in the step size. This suggests that the size of the nodes in the data structures can be as important as their traversal pattern. Next, we created a set of data-structures with different node sizes (ranging from 256 Bytes to 4096 Bytes), and traversed them linearly using a single computation loop. Delivered bandwidth for different addressing schemes

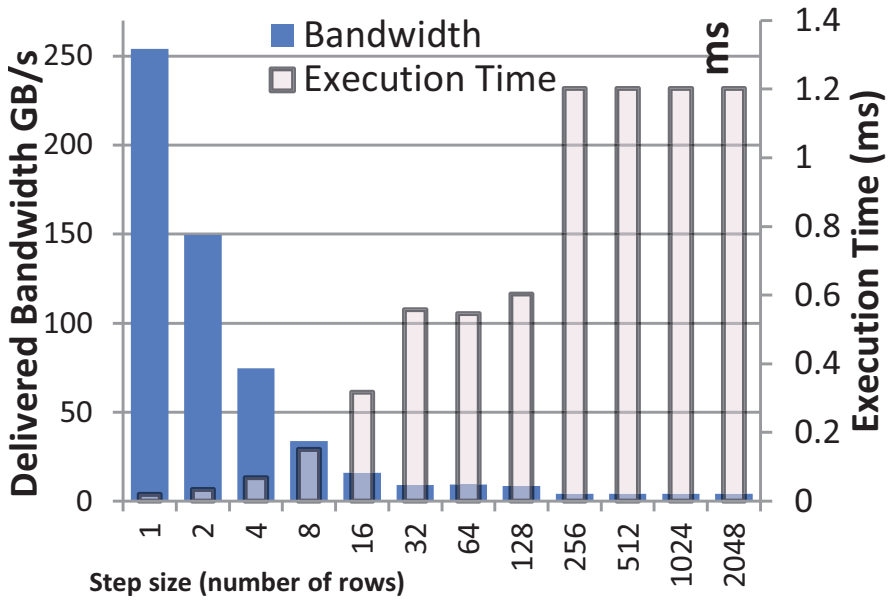


Figure 4.16: Effect of changing the step size of synthetic linear traffic on delivered bandwidth and execution time.

is plotted in Figure 4.17.a. As can be seen, *LOW.INTLV*'s bandwidth drops severely with the increase in the walk-step, while all scrambling mechanisms seem to be robust and insensitive of the walk-step.

In Figure 4.17.b, a random walk has been performed on the same data structures. Again, the *LOW.INTLV*'s bandwidth drops with a similar trend. These two experiments suggest that, regardless of the pattern of traversing a data-structure, size of its nodes has a crucial impact on the bandwidth extracted from HMC in the baseline *LOW.INTLV* addressing scheme. In particular, data structures composed of "fat" nodes with sizes equal to powers of two, seem to be dangerous to low-interleaved addressing, no matter how one traverses them. We would like to remind that fat structures are not uncommon. Some example includes descriptors in computer vision [142], and nodes in file-systems and databases [143].

To illustrate this issue better, Figure 4.18 plots the bank-access heat-map for all 256 DRAM banks present in HMC over time, where a linear walk has been performed on the data structure elements. Three addressing schemes are used in this plot: *LOW.INTLV*, *S.14.8.00*, and *S.17.5.04.T*. It can be clearly seen that, many of the DRAM banks remain unused for *LOW.INTLV* and more bank and vault conflicts happen with the increase in the walk step. This is issue does not occur in *S.17.5.04.T*. Also, it can be seen that *S.14.8.00* is not able to evenly distributing the memory accesses and still

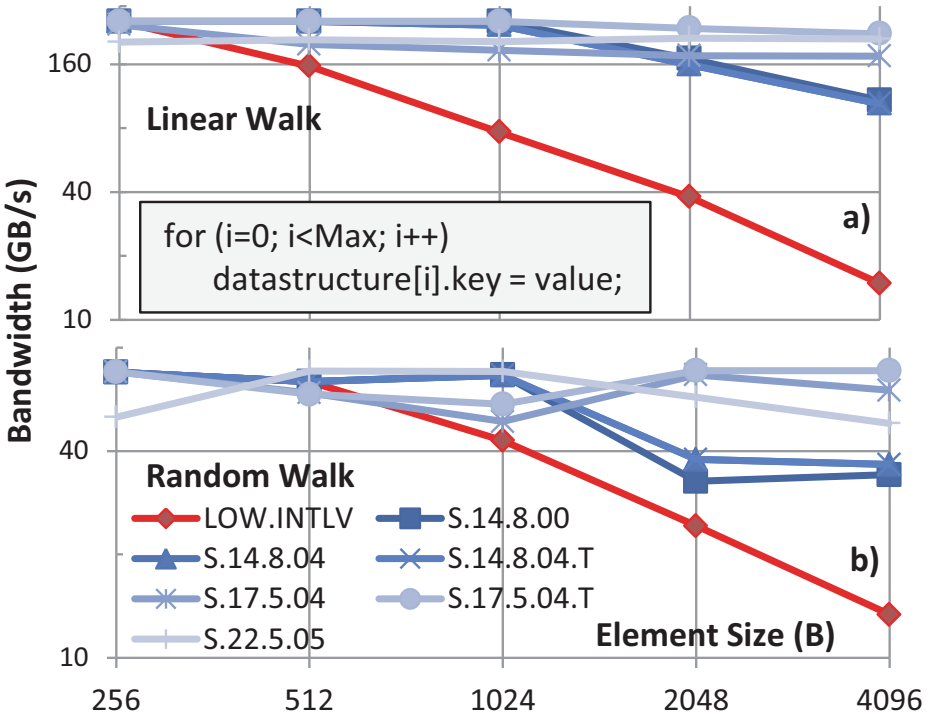


Figure 4.17: Delivered bandwidth compared between different addressing schemes, compared for linear walk (a), and random walk (b). The nodes size of the data structure has been changed from 256 Bytes to 4096 Bytes.

suffers from many bank conflicts. For this reason deeper scramblers like *S.17.5.04.T* are more desirable.

Similar plots have been shown in Figure 4.19 for three representative benchmarks: blackscholes, dense-matrix-add, and streamcluster. In all three cases, the address scrambling scheme evenly distributes the traffic over the banks and reduces bank and vault conflicts, and performs better than the conventional low-interleaving. The worst pattern belongs to blackscholes which is a numerical solver for Partial Differential Equations (PDE) [101].

One last experiment is performed to show another possibly inconsistent traffic pattern with low-interleaving. A medium sized dense matrix (512 x 512 x UInt64) is traversed once in the Row-Major (RM) order and then in Column Major (CM) order. The delivered bandwidth under different addressing schemes is plotted in Figure 4.20. RM traversal is insensitive to the underlying addressing scheme, since it is simply a linear walk with small walk steps. However, CM traversal is highly sensitive to the addressing scheme and its bandwidth drops severely with *LOW.INTLV* addressing.

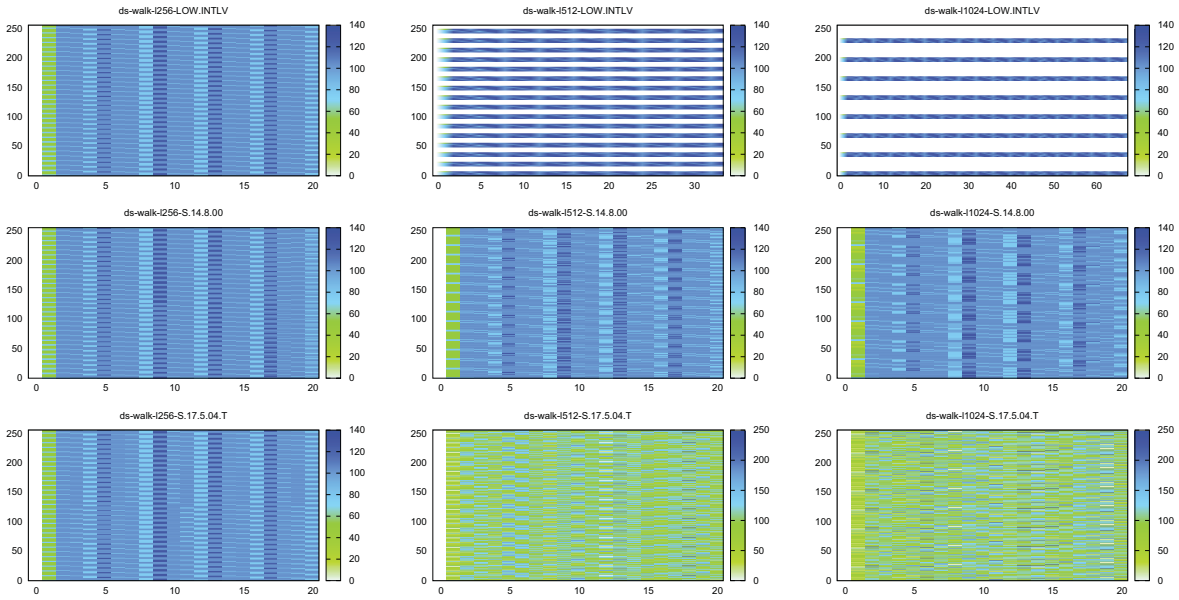


Figure 4.18: Heat map for the DRAM banks plotted over time, for linear walk on data structure nodes with different node sizes. In the upper plots address mapping is *LOW.INTLV*. In the middle plots the addressing scheme is S.14.8.00, and in the bottom ones S.17.5.04.T is utilized.

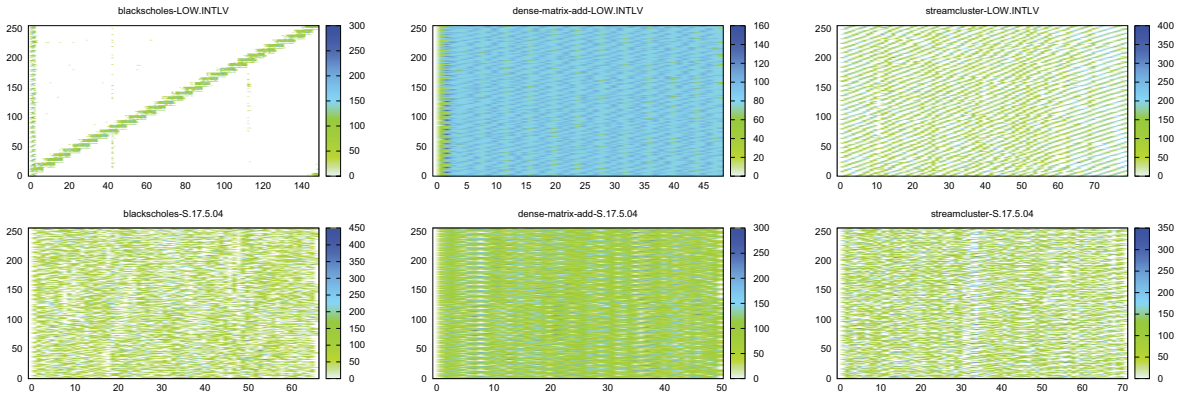


Figure 4.19: Heat map for the DRAM banks plotted over time, for three benchmarks: blackscholes, dense-matrix-add, and streamcluster. In the upper plots address mapping is *LOW.INTLV*, and in the bottom ones address mapping is S.17.5.04.

Again, size of the matrix and being aligned to powers of 2 have an important impact on the obtained results. Also, it should be noted that CM traversal is not uncommon

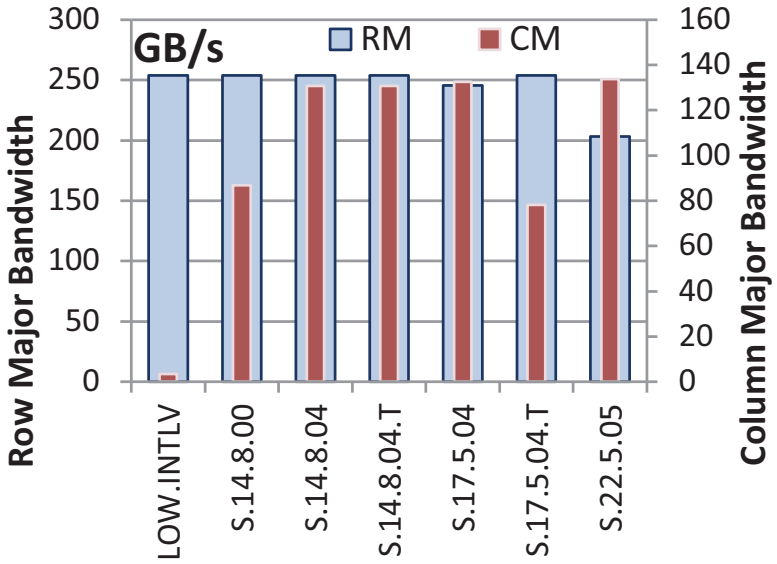


Figure 4.20: Effect of Row-major (RM) and Column-Major (CM) matrix traversal methods on delivered bandwidth.

in linear algebra and matrix transposition is a very simple example requiring it.

These experiments demonstrate that the default addressing scheme of HMC is not robust enough to be trusted for all traffic patterns. We would like to remind that the goal behind the design of HMC has been to abstract away the internal details of the memory system and hide them from the host processors. With this goal, we believe a more robust addressing methodology similar to the proposed scrambling schemes should be utilized to be able to serve a wide range of traffic patterns efficiently. Finally, the address scrambler illustrated in Figure 4.6 was synthesized using the same technology library and its combinational delay was found to be less than 0.6ns for an area of 600 Gate Equivalent (GE). This shows that the overhead of address scrambling is very small. Therefore it is sensible for each master port to have an instance of the address scrambler. The power consumption associated with the address scramblers were found to be negligible, as well.

4.5.3 Handling PIM Traffic

As stated earlier, the proposed architecture allows for integration of PIM devices by simply connecting them to the main crossbar switch. Yet, the main concern is the side-effects of PIM traffic on the traffics from the main links. Assuming a dual-port PIM device is connected through two additional links (we study the effect of number of ports later in this section), we aim to find the upper bound on the bandwidth that

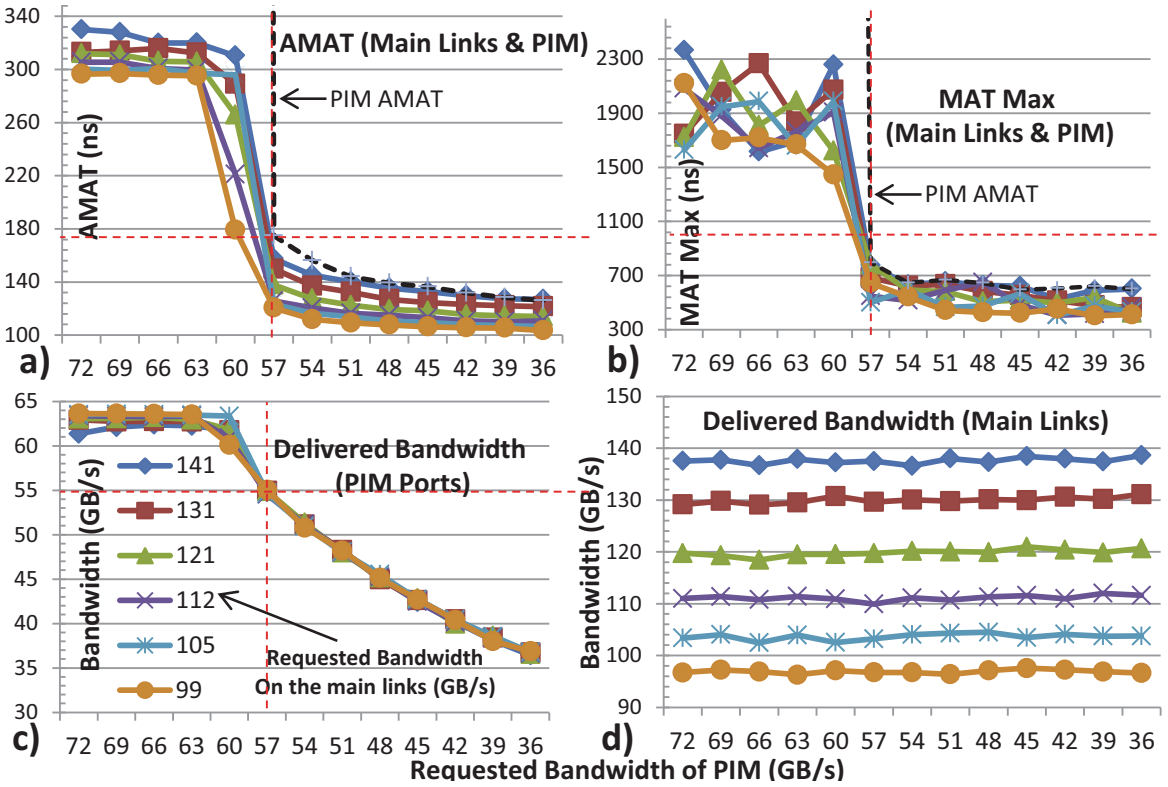


Figure 4.21: Increase in average (a) and maximum (b) memory access time caused by PIM. Delivered bandwidth to PIM as a function of requested bandwidth on PIM port (c), Drop in main bandwidth caused by interference of PIM (d)

it can request without disrupting the main traffic. We assume that PIM can process received data instantaneously, and on the main links as well as the PIM link we inject synthetic random transactions with various bandwidth profiles. Figure 4.21.a,b illustrate the average and maximum Memory Access Time (MAT), respectively. Figure 4.21.c shows delivered bandwidth to the PIM device, and Figure 4.21.d shows total delivered bandwidth on the main links. All plots have been characterized based on the amount of requested bandwidth on the main links (from 99GB/s to 141GB/s), and the X-axis shows requested bandwidth by the PIM device. It can be seen that, a dual-port PIM device can request up to its theoretical limit (64 GB/s) without pushing the system into saturation, and without observing any drop in the bandwidth of the main links. However, to keep the AMAT of the main links below 200ns (3X of zero load AMAT), and avoid saturation of the PIM links, more than 55 GB/s should not be delivered to PIM. This leads to an aggregate bandwidth delivery of 195 GB/s which is 95% of the 205 GB/s limit found in subsection 4.5.1. It is also worth mentioning

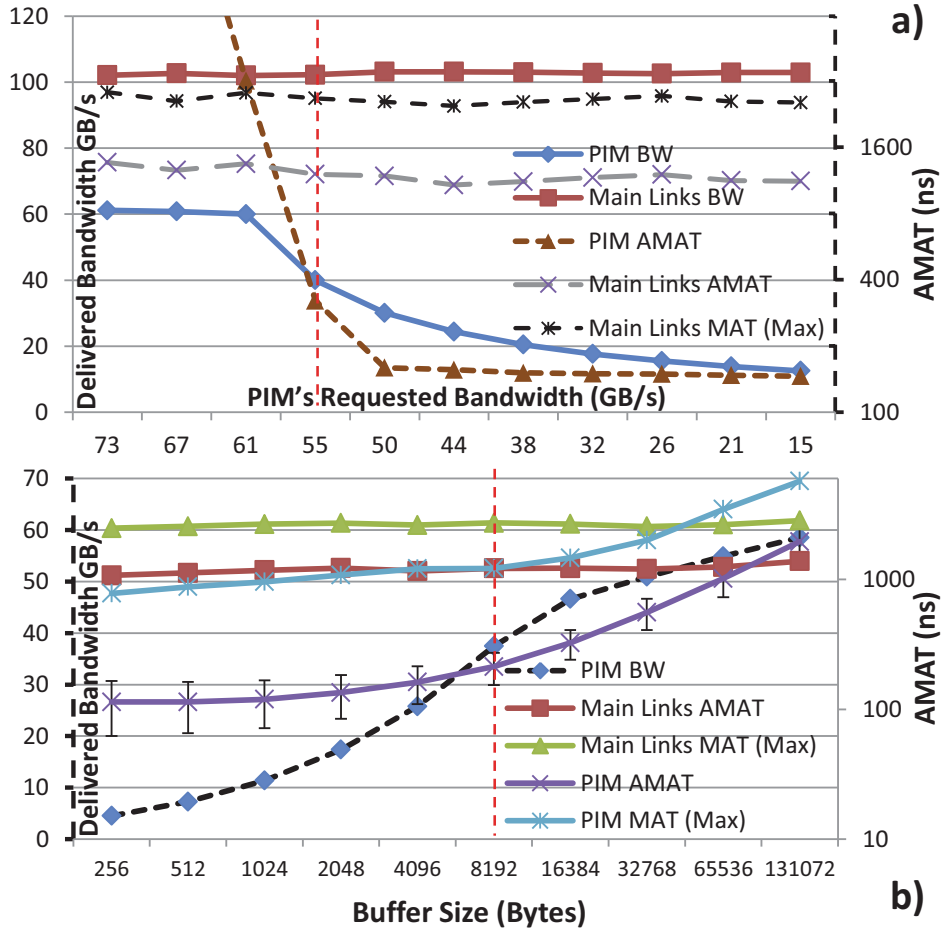


Figure 4.22: Effect of requested bandwidth by ideal PIM on x264 (a), and effect of buffer size of double-buffering PIM on x264 (b).

that, Average Memory Access Time (AMAT) is always below 350ns (5X of zero load AMAT). Since for typical general purpose servers, the traffic on the main links mostly consists of cache refill requests, they are very sensitive to latency. Therefore not only AMAT but also the worst-case MAT should be small. This also, limits the bandwidth delivered to PIM to 55 GB/s. [1]

To demonstrate this fact better, instead of random traffic, real traces of the high bandwidth demanding x264 benchmark with time compression are applied on the main port, and interference caused by random traffic from a PIM device with two ports has been plotted in Figure 4.22.a. This plot again shows that the full bandwidth delivered to the dual-port PIM does not cause any disruption to the bandwidth or AMAT of the main links. Also, the PIM port can receive 40 GB/s without saturating itself with an

[1] A less conservative bandwidth partitioning could be speculated for accelerator-dominated traffics (e.g. GPGPU) which are much less latency-sensitive.

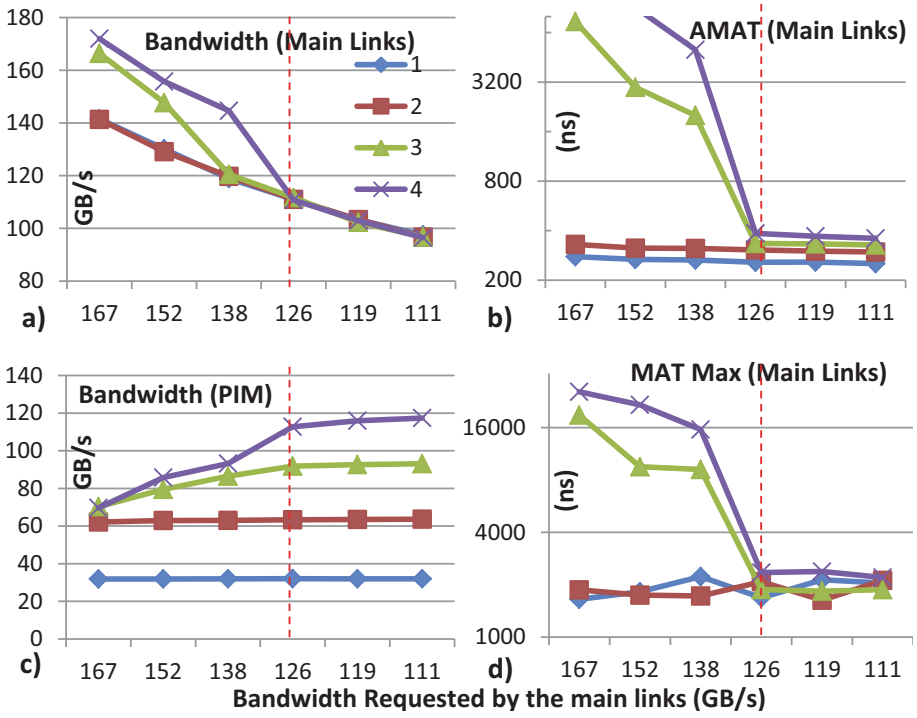


Figure 4.23: Effect of increasing the number of PIM ports on delivered bandwidth (a) to the main links, AMAT of the main links (b), PIM's delivered bandwidth (c), maximum MAT (b)

increased AMAT of below 400ns (5X of zero load AMAT). Execution time increase in the x264 application was found to be less than 5%.

Next, we assume that PIM uses two Direct Memory Access (DMA) engines each working with double-buffering mechanism, working on one buffer, while the other one is being fetched from the memory. Again, x264 is played on the main links. Results are shown in Figure 4.22.b. This plot identifies the maximum amount of time that PIM can spend computing on the data, to effectively hide the DRAM access latency. Buffer size larger than 8KB does does not seem to be desirable due to super-linear increase in the AMAT of the PIM link itself.

Figure 4.23 shows the maximum bandwidth that a multi-ported PIM can request based on different bandwidth demands by the main links. The requested bandwidth on the main links has been characterized from 111 GB/s to 167 GB/s and PIM's number of ports has been increased up to 4. We can see that, when host is demanding less than 120 GB/s, PIM can use the remaining bandwidth safely without pushing the system into saturation. Also, a dual-port PIM can request up to 64 GB/s without any disrupt to the main links regardless of the demand on the main links, while beyond 2 ports

careful bandwidth throttling is required. This is because, AMAT on the main links becomes more sensitive to PIM's traffic as the number of its ports increases.

4.6 Summary

In this chapter, we presented a high performance AXI-compatible interconnect to support near memory computation on the LoB die of SMC. Cycle accurate simulation results demonstrated that, the proposed interconnect can easily meet the demands of current and future projections of HMC (Up to 205 GB/s READ bandwidth with 4 serial links and 32 memory vaults). Moreover, the interference between the PIM traffic and the main links was found to be negligible when PIM has up to 2 ports, and 64 GB/s bandwidth can be delivered to it without any major disrupt on the main links. It was shown that low-interleaved addressing is not reliable enough for an abstracted memory such as HMC. Fat data structures with power-of-two node sizes were particularly identified as unfavorable patterns for low-interleaving. A more robust address scrambling mechanism was proposed and it was shown to effectively reduce bank/vault conflicts. Finally, logic synthesis confirms that our proposed models are implementable and effective in terms of power, area, timing (power consumption less than 5mW up to 1 GHz and area less than $0.4mm^2$).

In chapter 5 we design a high-level full-system simulation model based on the parameters and calibration results obtained in this chapter. We will design an actual processor in memory architecture and evaluate it in a full system context in presence of all offloading and dynamic overheads.

Chapter 5

Processor-in-Memory Design for the Smart Memory Cube

In chapter 4 we introduced SMC's concept and evaluated its architectural implications and requirements using cycle accurate models and different traffic patterns. In this chapter we take one step further and present the first exploration steps towards design of a Processor-in-Memory (PIM) architecture for SMC. An accurate simulation environment is developed, along with a full featured software stack. All offloading and dynamic overheads caused by the operating system, cache coherence, and memory management are considered. Also, a zero-copy pointer passing mechanism has been devised, to allow low overhead data sharing between the host and the PIM.

5.1 Motivations and Challenges

Near memory computation can provide two main opportunities: (1) reduction in data movement by vicinity to the main storage resulting in reduced memory access latency and energy, (2) higher bandwidth provided by Through Silicon Vias (TSVs) in comparison with the interface to the host which is limited to the pins. Most recent works exploit the second opportunity by trying to accelerate data-intensive applications with large bandwidth demands ([51][15][122]). In [50] and [53] also, networks of 3D stacked memories are formed and host processors are attached to their peripheries, providing even more hospitality for processing-in-memory (PIM) due to huge bandwidth internal to the memory-centric network. These platforms, however, are highly costly and suitable for high-end products with extremely high performance goals [53]. Also, a look at the latest HMC Specification [13] reveals that its ultra-fast serial interface is able to

deliver as much bandwidth as is available in the 3D stack. Four serial links each with 32 lanes operating from 12.5Gb/s to 30Gb/s serve for this purpose [13]. For this reason, the same bandwidth available to a PIM on the logic die is also theoretically available to the external host, and high-performance processing clusters or GPU architectures executing highly parallel and optimized applications can demand and exploit this huge bandwidth [144][145]. This puts PIM in a difficult but realistic position with its main obvious advantage over the external world being vicinity to the memory (lower access latency and energy) and not an increased memory bandwidth.

In this chapter, we focus on this dark corner of the PIM research, and try to demonstrate that even if delivered bandwidth to the host can be as high as the internal bandwidth of the memory, PIM's vicinity to memory itself can provide interesting opportunities for energy and performance optimization. We focus on a worst-case scenario where a single PIM processor is trying to compete with a single thread on host. In our experiments caches are not thrashed, the memory interface is not saturated, and the host can demand as much bandwidth as it requires.

As explained in chapter 4, our PIM proposal (called the Smart Memory Cube) is built on top of the existing HMC standard with full compatibility with its IO interface specification. We have developed a full-system simulation environment called SMCSim and verified its accuracy against the Cycle-Accurate (CA) model described in chapter 4. SMCSim models the complete software and hardware stack ranging from high-level user application to low-level firmware and hardware layers. It takes into account the offloading and dynamic overheads caused by the operating system, cache coherence, and memory management, as well. We devised an optimized memory virtualization scheme for zero-copy data sharing between host and PIM; enhanced PIM's operations by the aid from atomic in-memory operations; and improved PIM's memory access by means of a flexible Direct Memory Access (DMA) engine. Our proposal is not dependent on the ISA of the host processors and provides high flexibility and programmability by means of a full-featured software stack. After presenting the related works in this area, in section 5.3 and section 5.4 the SMCSim environment is introduced and design of the PIM is described. Next, the software stack is presented in section 5.5. Finally, experimental results are presented and a summary of the obtained results and conclusions are given in Sections section 5.6 and section 5.7.

5.2 Related Works

The design space for near memory computation is enormous, and several different concerns need to be addressed such as the micro-architecture of the PIM processor, support for memory management and cache coherence, and communication mechanisms with the host processors [146].In [49] a CGRA is located on a separate die and connected to the DRAM dies through Through Silicon Vias (TSVs). Segmented memory without caching is used and 46% energy saving along with 1.6X performance gain for Big Data applications are achieved. In [15], 64 Cortex-A5 processors form a PIM cluster. Memory is preallocated in contiguous regions, and two levels of caches with on demand software flushing are available. For Big Data workloads 1.1X and 23% and performance and energy gain are reported. The main difference between our work in this chapter and these two papers is flexible support for virtual memory as well as considering the offloading overheads in our analysis.

Active Memory Cube (AMC) [50] extends the logic layer of the HMC with clusters of vector processors without caches. Hardware coherence is maintained with the host and virtual memory support has been provided. 2X performance improvement has been reported for dense matrix operations, increasing to 5X when vicinity aware memory allocation schemes are utilized. In [51], PIM is comprised of CPUs and GPUs. Memory is preallocated to the PIM, and analytical evaluations show 85% energy saving with minor performance improvements for graph, HPC, and GPGPU workloads. [52] augment the logic die with 16 light-weight general purpose cores with 2 levels of caches, hardware cache coherence, and preallocated memory. For scientific applications up to 2X performance gain has been reported. Tesseract [53] features a network of memory cubes each accommodating 32 in-order cores with L1 caches and two prefetchers, optimized for parallelizing the PageRank algorithm. Uncacheable regions are shared with PIM and segmented memory without paging is supported. Up to 10X performance improvement and 87% energy reduction has been provided in comparison with high-performance server hosts. Unlike these three works, we focus on a context which external memory interface is not bandwidth saturated and PIM's benefits are not determined by the delivered bandwidth. In addition, we utilize atomic commands and consider the offloading overheads in our studies, as well.

In [122], the memory stack has been augmented with low level "atomic" in-memory operations. Host instructions are augmented, and full virtual memory support and hardware cache coherence is provided. for Big Data workloads up to 20% performance

and 1.6X energy gain is obtained. The main difference between our proposal and this work is that our PIM supports flexible execution of different computation kernels and its acceleration is not limited to the atomic operations only. Moreover, our solution is not dependent on the ISA of the host and with a proper software stack any host platform can communicate with it through its memory-mapped interface. Lastly, in [121] up to 15X performance gain, and in [124] up to two orders of magnitude energy and performance gain compared to the host are reported. However, use of open-loop trace-based simulation, without considering the feedback effects and dynamic overheads makes their results highly optimistic.

In this chapter we present design and exploration of a PIM architecture for SMC. The goal is to provide flexible computational capabilities by means of full virtual memory support and a full-featured software stack compatible with the Parallel Programming Application Programming Interfaces (API). This is the first PIM effort to accurately model all layers from high-level user applications, to low level drivers, operating system (OS), and hardware, considering all dynamic overheads related to the OS, caches, and memory management. Besides, a comprehensive accuracy verification versus a cycle-accurate model has been performed which improves the quality of the obtained results.

5.3 The SMC Simulation Environment

SMCSim is a high-level simulation environment developed based on gem5 [100], capable of modeling an SMC device attached to a complete host System on Chip (SoC). Figure 5.1 illustrates an example of one such platform modeled in this environment. SMCSim has been designed based on gem5's General Memory System model [147] exploiting its flexibility, modularity, and high simulation speed, as well as, features such as check-pointing and dynamic CPU switching.

Figure 5.1 highlights the most important components in this environment: The host is an ARMv7 SoC capable of booting a full-featured operating system. Caches and interconnects are adopted from gem5, as well, without modifications. Inside the SMC, the vault controllers and the main interconnect are modeled using pre-existing components and tuned based on the CA model. A PIM is located on the LoB layer with flexible and generic computational capabilities. This configuration is completely consistent with the current release of the HMC Specification [13]. Also, SMC is not dependent or limited to any ISA and it is exposed to the host via memory mapped

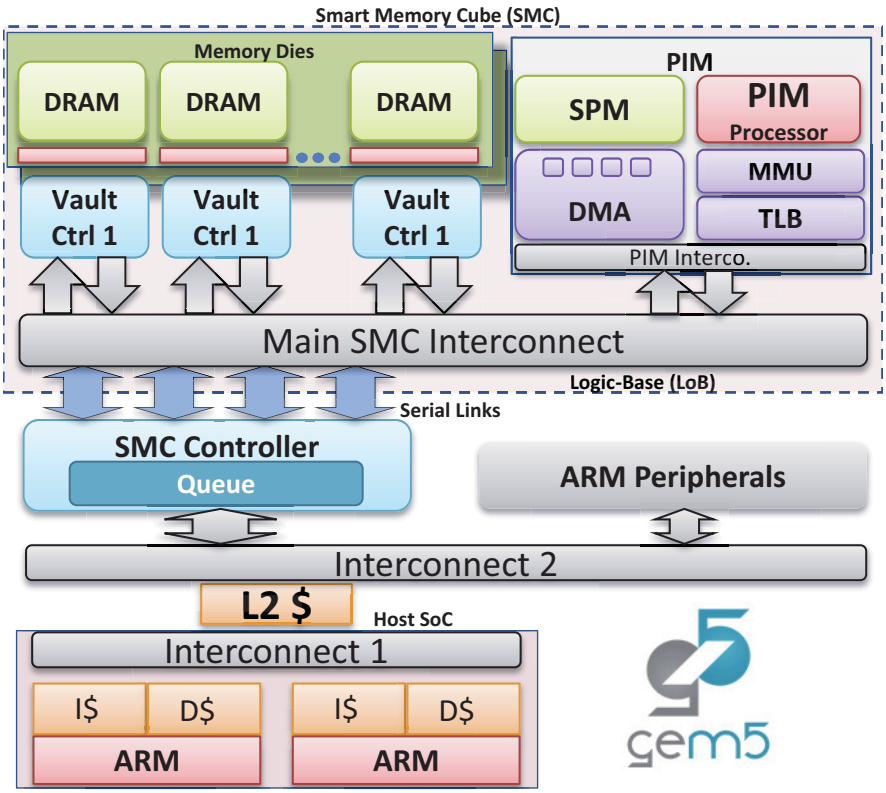


Figure 5.1: An overview of the SMCSim Environment for Design Space Exploration of the Smart Memory Cube (SMC)

regions. Therefore, any host platform should be able to communicate with it by the aid from a proper software stack. For the serial links: bandwidth, serialization latency, and packetization overheads are obtained from [13]. SMC/HMC controller, in general, is responsible for translating the host protocol (AXI for example) to the serial links protocol. Plus, it should have large internal buffers to hide the access latency of the cube. More advanced global scheduling policies and reordering and steering of transactions can be implemented if required [148]. However in our simulation environment, the SMC Controller simply queues the incoming transactions and schedules them to serial links using a simple round-robin mechanism to balance the load among them. Serial links accept the same address ranges and each packet can travel over any of them [140][13]. Different sources of latency are modeled in this environment and calibrated in subsection 5.6.1.

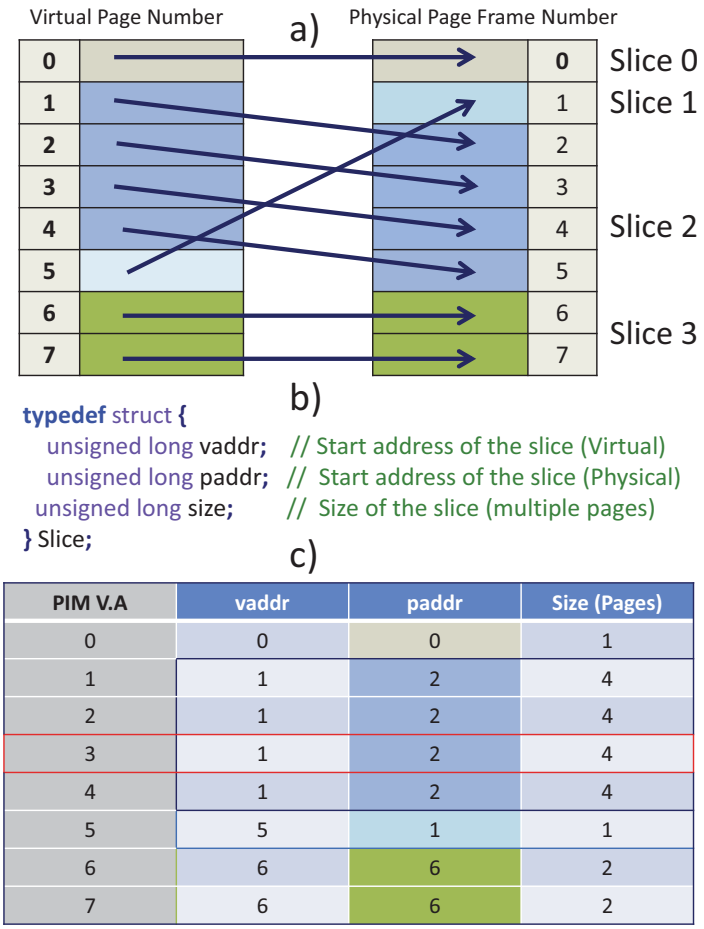
5.4 Design of the Processor-in-Memory

We have chosen an ARM Cortex-A15 core without caches or prefetchers (See subsection 5.4.2 for motivation and reasoning), and augmented it with low cost components to enhance its capabilities as a Processor-in-Memory. Our choice of ARM is because it offers a mature software stack, its system bridges (AXI) are well understood, and it is an energy-efficient architecture. Nevertheless, the architecture is not limited to it. As shown in Figure 5.1, PIM is attached to the main interconnect on the LoB through its own local interconnect, and features a Scratchpad Memory (SPM), a Direct Memory Access (DMA) engine, a Translation Look-aside Buffer (TLB) along with a Memory Management Unit (MMU). In this section we will describe the role of these components.

5.4.1 PIM's Memory Model

PIM has been designed to directly access user-space virtual memory. The TLB illustrated in Figure 5.1 serves for this purpose. Apart from memory protection benefits, this replaces memory-copy from user's memory to PIM with a simple virtual pointer passing. Scalability and programmability are improved, and offloading overheads are reduced to a great extent. Since user's memory is paged in conventional architectures, PIM should support this, as well. To add more flexibility we introduce the concept of *slices* as a generalization to memory pages: *slice* is a region of memory composed of 1 or more memory-pages which is contiguous in both virtual and physical memory spaces. With this definition, contiguous memory pages which map to contiguous page-frames can be merged to build larger slices, with arbitrary sizes.

PIM's memory management is done at the granularity of the slices. The first slice is devoted to PIM's scratchpad memory, and the rest of the memory space is mapped immediately after this region. Upon a TLB miss, a data structure in DRAM called the *slice-table* is consulted and the translation rules are updated on a Least Recently Used (LRU) basis. *Slice-table* is similar to page-tables and contains all translation rules for the computation kernel currently executing on PIM. It is built during the task offloading procedure by PIM's device-driver (section 5.5). Since the slices can have arbitrary sizes, implementing the slice-table as a simple table of slices can complicate the lookup procedure. In order to minimize the number of DRAM access required to fetch rules from this table, in this table, we keep entries for the underlying pages rather than the slices. Therefore, for pages in the same slice, we store the same translation



PIM V.A	vaddr	paddr	Size (Pages)
0	0	0	1
1	1	2	4
2	1	2	4
3	1	2	4
4	1	2	4
5	5	1	1
6	6	6	2
7	6	6	2

Figure 5.2: A sample page to frame mapping highlighting the merged pages (a), The *slice-table* data structure (b), values stored in the *slice-table* for current example (c)

rule (See Figure 5.2). This simplifies the lookup procedure at the cost of redundant rules in the slice-table. For example if PIM accesses the virtual address 3 (highlighted in Figure 5.2.c), using a single DRAM burst of 3 words (12Bytes for 32bit architectures) it is possible to fetch the rule completely for Slice 2. This way, there is no need to access the slice-table twice, regardless of the size of the slices.

Most host-side accelerators with virtual memory support rely on the host processor to refill the rules in their TLB. The OS consults its page-table to reprogram the IO-MMU of the device, and then wakes up the accelerator to continue its operation. Since PIM is far from the host processors, asking them for a refill upon every miss can result in a large delay. As an alternative, PIM's TLB contains a simple controller responsible for fetching the required rule from the slice-table. Apart from the performance benefits, this allows for fully independent execution of PIM.

5.4.2 Enhancing PIM's Functionality

A simple zero-load latency analysis (subsection 5.6.1) reveals that the latency of directly accessing DRAM by PIM is harmful to its performance, therefore latency-hiding mechanisms are required. Caches and prefetchers provide a higher-level of abstraction without much control. This is desired for SoCs far from the memory and flexible enough to support different main memory configurations. DMA engines, on the other hand, provide more control, making more sense in a near-memory processor. We have augmented PIM with a DMA engine capable of bulk data transfers between the DRAM vaults and its SPM (See Figure 5.1). It allows multiple outstanding transactions by having several DMA resources, and accepts virtual address ranges without any alignment or size restrictions, on top of the functionalities provided by gem5's DMA component.

A complementary way to address this problem is to move some very specific arithmetic operations directly to the DRAM dies and ask the vault controller to do them "atomically". Abstracted memory interface provided by the HMC facilitates implementation of these operations and the atomic HMC commands [13] are good examples but for synchronization purposes. In-memory operations can reduce data movement when computation is local to one DRAM row [122]. We have augmented our vault controllers with three types of atomic commands suitable for the benchmarks under our study. On the other side, instead of modifying PIMs ISA to support these commands, we added specific memory mapped registers to the PIM processor. By configuring these registers PIM is able to send atomic operations towards the SMC vaults. Here is a list of the implemented atomic commands: (See subsection 5.6.2 for their usage)

- *atomic-increment*: interprets the value stored in the intended address in DRAM as an integer and increments it by 1.
- *atomic-min*: sends an *Immediate* value to the intended DRAM location. If *Immediate* is less than the value currently available in that address, the value will be replaced with *Immediate*, otherwise it will be left untouched.
- *atomic-add-immediate*: interprets the value stored in the intended address as a single precision floating-point and add the *Immediate* value to it.

These commands can be implemented either in the vault controllers, or in the DRAM dies due to their low computational complexity. Also, their execution latency is very low, so they can be easily hidden behind DRAM timings.

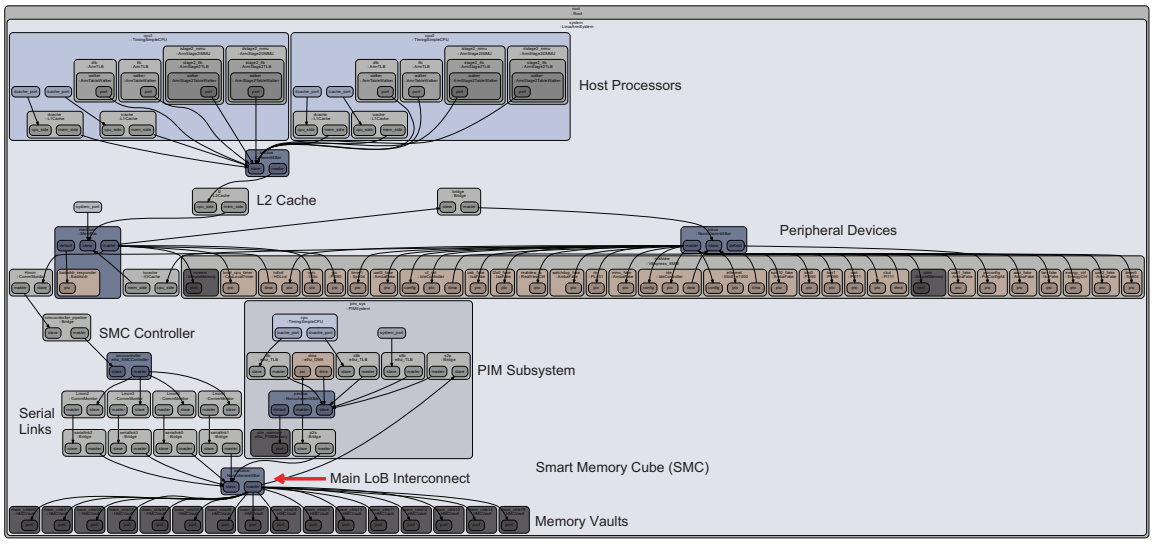


Figure 5.3: Visualization of the simulated system in SMCSim composed of the host processors, the interconnects, SMC controller and the serial links, the SMC model, and the PIM subsystem.

For computations that need to gather information not fully localized to a single memory vault, DMA can be more beneficial. While, highly localized computations with low computational intensity are better performed as close as possible to the memory dies, by means of the atomic commands. A visualization of the whole simulated system in gem5 is illustrated in Figure 5.3.

5.5 Design of the Software Stack

A software-stack has been developed for the user level applications to view PIM as a standard accelerator (See Figure 5.4). At the lowest level, a resident program runs on PIM performing the required tasks. A dynamic binary offloading mechanism has been designed to modify this code during runtime. PIM also features a set of configuration registers (Figure 5.4) mapped in the physical address space and accessible by the host. Plus a few kilobytes of scratchpad space is provided supporting flexible DMA transfers. PIM's device driver has been adopted from Mali GPU's driver [149] and is compatible with standard accelerators as well as parallel programming APIs such as OpenCL. This light-weight driver provides a low-overhead and high-performance communication mechanism between the API and PIM. An object-oriented user-level API has been provided, as well, to abstract away the details of the device driver and to facilitate

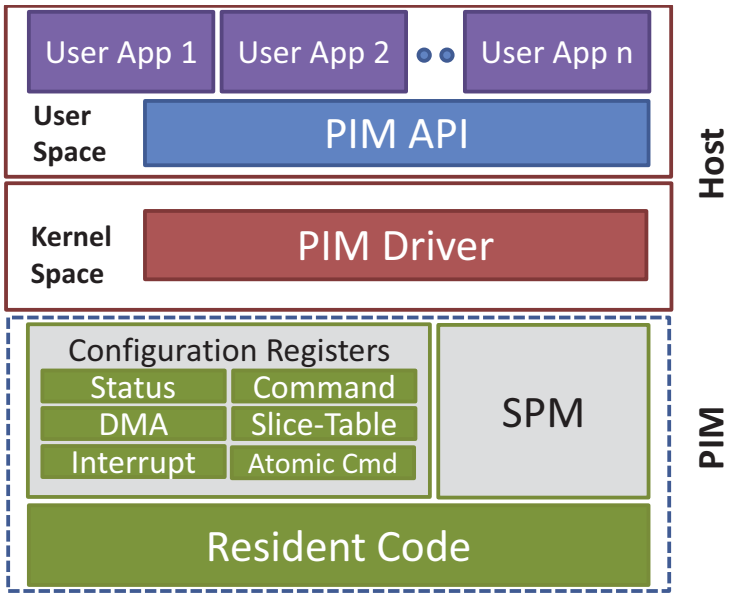


Figure 5.4: PIM's software stack

user's interface. Offloading and coordinating the computations on PIM are initiated by this API.

5.5.1 Offloading Mechanisms

PIM targets execution of medium sized computation kernels having less than a few kilobytes of instructions. The host processor parses the binary Executable and Linkable Format (ELF) file related to a precompiled computation kernel, and dynamically offloads *.text* and *.rodata* sections to PIM's memory map by the aid from the API. This procedure is called the *kernel-offloading*. On the other hand, the virtual pointer to pre-allocated user level data structures need to be sent to PIM for the actual execution to take place (*task-offloading*). PIM's API sends the page numbers associated with user data structures to the driver, and the driver builds the slice-table in the kernel memory space using the physical addresses. Next, caches are flushed and the virtual pointers are written to PIM's memory mapped registers. An interrupt is then sent to PIM to wake it up for execution. This mechanism prevents PIM from accessing unwanted physical memory locations, and allows it to traverse user-level data structures without any effort. A polling thread in PIM's API waits for completion of the offloaded task.

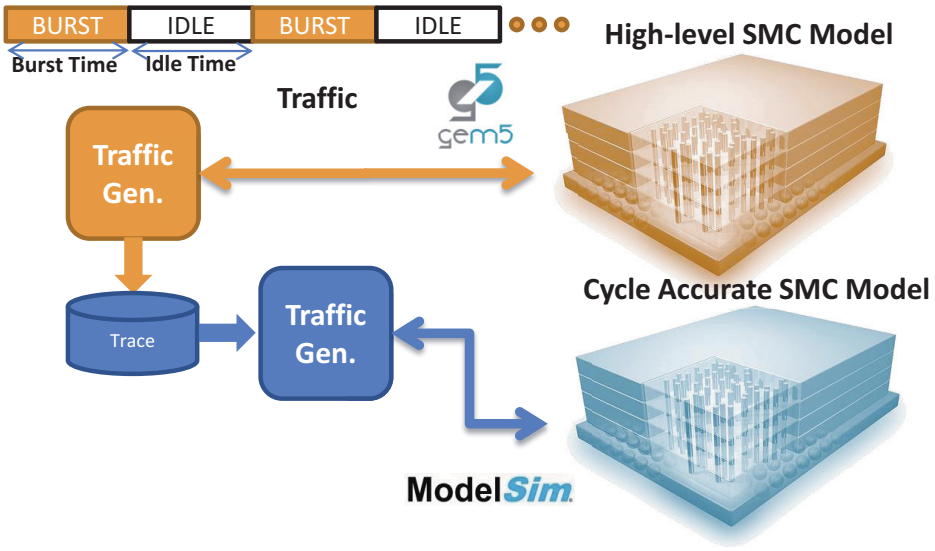


Figure 5.5: An overview of the accuracy comparison methodology between the high-level SMC model developed in gem5 and the cycle-accurate SMC model.

5.6 Experimental Results

Our baseline host system is composed of two Cortex-A15 CPU cores @2GHz with 32KB of instruction cache, 64KB of data cache, and a shared 2MB L2 cache with associativity of 8 as the last-level cache (LLC). The block size of all caches is increased to 256B to match the row buffer size of the HMC model (effect of cache block size is studied in subsection 5.6.2). The memory cube model provides 512MB of memory with 16 vaults, 4 stacked memory dies, and 2 banks per partition. PIM has a single core processor similar to the host processors running at the same frequency, with the possibility of voltage and frequency scaling by means of dedicated clock and voltage domains on the LoB. Maximum burst size of PIM's DMA has been set to 256B by default.

5.6.1 Accuracy Verification and Calibration

For the high-level simulation results to be trusted, verification is required against a CA simulation model. We performed a detailed accuracy comparison of the high-level SMC against the CA model presented in chapter 4. We applied identical traffic patterns with various bandwidth demands to both CA and gem5-based models, and compared their delivered bandwidth and total execution time over a large design space defined by the architectural parameter. An overview of this procedure is illustrated in Figure 5.5.

Table 5.1: Zero-load latency of memory accesses

HOST: L2 cache refill latency - Size:256Bytes [Total: 102.3ns]		
Membus	1Cycle@2GHz	(FlitSize=64b)
SMCController	8Cycles@2GHz	(Pipeline Latency) [150]
SERDES	1.6ns	(SER=1.6 DES=1.6) [141]
Packet xfer	13.6ns	(128b hdr 16x10Gbits/s) [13]
PCB Trace	3.2ns + 3.2ns	(Round Trip)
SMCXBar	1Cycle@1GHz	(FlitSize=256b)
VaultCtrl.frontend	4Cycles@1.2GHz	Request Processing [139]
tRCD	13.75ns	Activate [141]
tCL	13.75ns	Issue Read Command
tBURST	25.6ns	(For 256Bytes packet)
VaultCtrl.backend	4Cycles@1.2GHz	Response Processing [139]
SERDES	1.6ns	(SER=1.6 DES=1.6)
Packet Transfer	13.6ns	(128b hdr 16x10Gbits/s)
SMCController	1Cycles@2GHz	(Pipeline Latency)
Membus	1Cycle@2GHz	(FlitSize=64b)
PIM: Latency of a 4Bytes read access (No caches) [Total: 39.1ns]		
PIM Interco.	1Cycle@1GHz	(FlitSize=32b)
SMCXBar	1Cycle@1GHz	(FlitSize=256b)
VaultCtrl.frontend	4Cycles@1.2GHz	
tRCD	13.75ns	
tCL	13.75ns	
tBURST	3.2ns	(1 Beat only)
VaultCtrl.backend	4Cycles@1.2GHz	
PimBus	1Cycle@1GHz	

Several experiments demonstrated that total execution time and delivered bandwidth of the gem5-based model correlate well with the CA model: with low or medium traffic pressure, the difference was less than 1%, and for high pressure saturating traffic the difference was bounded by 5%, in all cases. A small subset of the accuracy comparison experiments is brought in Appendix A. Next, we calibrated the latency of the individual components based on the available data from the literature and the state-of-the-art. Results are shown in Table 5.1. As can be seen, the zero-load memory access latency of a cache refill request packet from the L2 cache port of the host to the SMC was estimated to be about 100ns, while a single-word memory read from PIM towards the memory vaults has a latency of about 40ns.

5.6.2 Benchmarking Results

Data intensive applications often categorized as "Big Data" workloads are widely chosen in the literature as the target for near memory acceleration [49][15][122][53]. The common trait shared by most of these applications is sensitivity to memory latency and high bandwidth demand. Graph traversal applications are an interesting example in this group due to their unpredictable memory access patterns and high ratio of memory access to computation [122]. A common use for these benchmarks is social network analysis. We have chosen four large-scale graph processing applications, and try to accelerate their main computing loop (i.e. the computation kernel) by offloading

it to PIM for execution. Here is the list of the benchmarks studied in this chapter: (The source codes of these kernels are brought in Appendix B)

Average Teenage Follower (ATF)

ATF counts for each vertex the number of its teenage followers by iterating over all teenager vertices and incrementing the "follower" counters of their successor vertices [122]. This generates a very large amount of random memory accesses over the entire graph, for this reason, we have implemented an *atomic-increment* command inside the memory cube for this purpose, which is able to increment a location in the DRAM using a single command.

Breadth-First Search (BFS)

BFS visits the vertices closer to a given source vertex first by means of a FIFO queue [122]. No atomic operations were utilized to accelerate this kernel, however, the queue for visiting the nodes has been implemented in PIM's SPM, given that in sparse graphs its size is determined by the maximum outage degree of the nodes and not the number of nodes.

PageRank (PR)

PR is a well-known algorithm that calculates the importance of vertices in a graph [53]. We have implemented a single-precision floating point version of this kernel, plus an atomic floating-point *add-immediate* on the vault side.

Bellman-Ford Shortest Path (BF)

BF finds the shortest path from a given source vertex to other vertices in a graph [122]. Vault controllers have been augmented with *atomic-min* to facilitate its execution.

As described before, in this chapter we focus on the single threaded version of these kernels, and study the performance and energy impact of offloading a single execution thread to PIM. This is a worst-case scenario to see if still PIM operates better than the host. It can be easily verified that the choice for underlying representation of the graphs is as important as the functionality of the kernel itself. While Adjacency-Matrix leads to simplest implementations, it is not scalable to millions of nodes and traversing adjacent nodes in large sparse graphs is costly. Compressed-Sparse-Row (CSR) [151] representation and List of Lists (LIL) [152][153] are more widely used and suitable

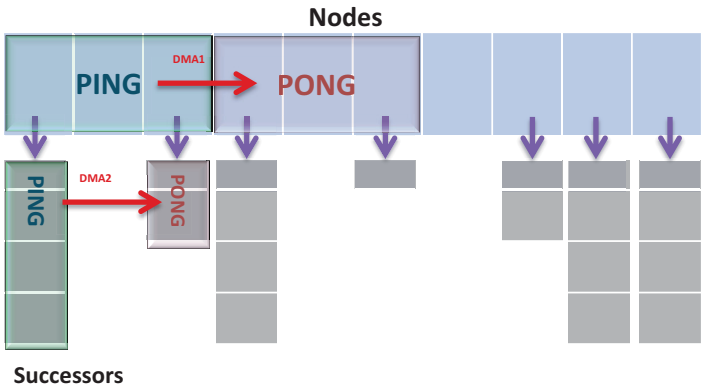


Figure 5.6: Traversing the sparse graphs represented using LIL format using two DMA resources

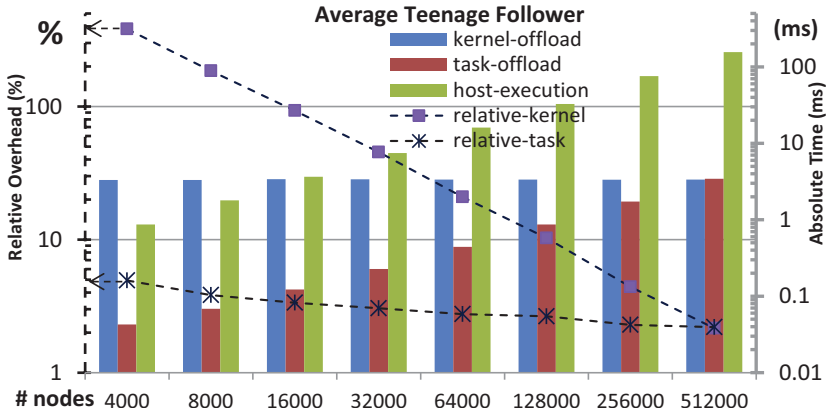


Figure 5.7: Offloading overheads in execution of the ATF kernel

for graph traversal benchmarks. We have chosen LIL as it is easily parallelizable and scalable to multiple processing nodes. We have used randomly generated sparse graphs ranging from 4K node to 512K nodes with characteristics obtained from real world data sets [154]. We use two DMA resources to efficiently hide the latency of traversing the LIL as illustrated in Figure 5.6.

First we study offloading overheads. ATF has been executed on the host side and then offloaded to PIM. In Figure 5.7, *kernel-offload* represents the overhead associated with reading the ELF file from the secondary storage of the host, parsing it, and offloading the binary code (as explained in subsection 5.5.1). *task-offload* is all overheads associated with virtual pointer passing to PIM for the graph to be analyzed, and *host-execution* is the absolute execution time of the kernel on host. It can be seen that the task-offload overhead decreases with the size of the graph and is always below 5% of the total execution time of the ATF. Most of this overhead is due to cache flushing and

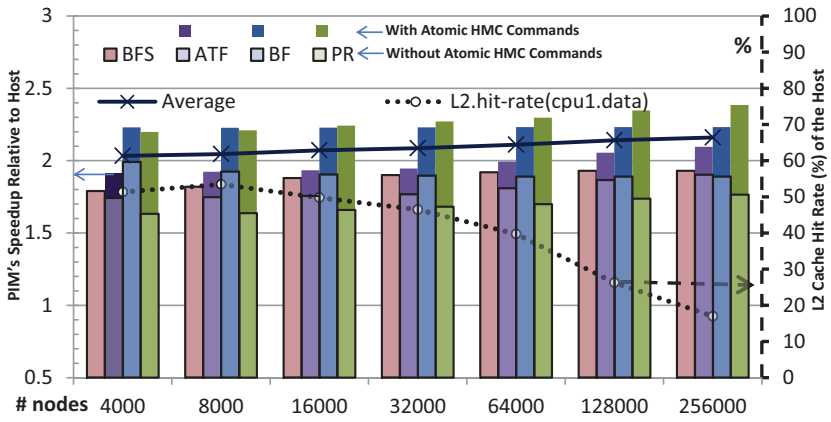


Figure 5.8: PIM's speed-up with/without SMC Atomic Commands (left axis), LLC hit-rate associated with the data port of the executing CPU (right axis)

less than 15% of it belongs to building the slice-table and pointer passing. It should be noted that, kernel-offload is usually performed once per several executions, therefore, its cost is amortized among them. Plus, for kernels like PR with several iterations relative overheads becomes even lower. Another point to mention is that, the overhead of the polling mechanism on the host to check for completion of the offloaded task was found to be negligible. This shows that an interrupt mechanism is not necessary and polling results in acceptable performance for execution of medium sized kernels. One should note that in current release of the HMC specifications [13] no interrupt mechanism from the cube towards the host is provisioned.

Next, we analyze the speed-up achieved by PIM in terms of host's execution time divided by PIM's. The number of graph nodes has been changed and Figure 5.8 (left vertical axis) illustrates the results. Lightly shaded columns represent PIM's execution without any aid from the atomic HMC commands, while the highly shaded ones use them. PIM's frequency is equal to the host (2 GHz). An average speed-up of 2X is observable across different graph sizes, and as the size increases, speed-up increases as well. This can be associated with increase in cache misses in the LLC of the host (plotted on the right vertical axis). Furthermore, the average benefit of using *atomic-increment*, *atomic-min*, *atomic-float-add* is obtained as 10%, 18%, and 35%, respectively.

A cache line size of 256 Bytes was mentioned before for both cache levels in the host. We can see in Figure 5.9 that this choice has been in favor of the host in terms of performance, leading to an increase in the LLC hit-rate. This is mainly due to sequential

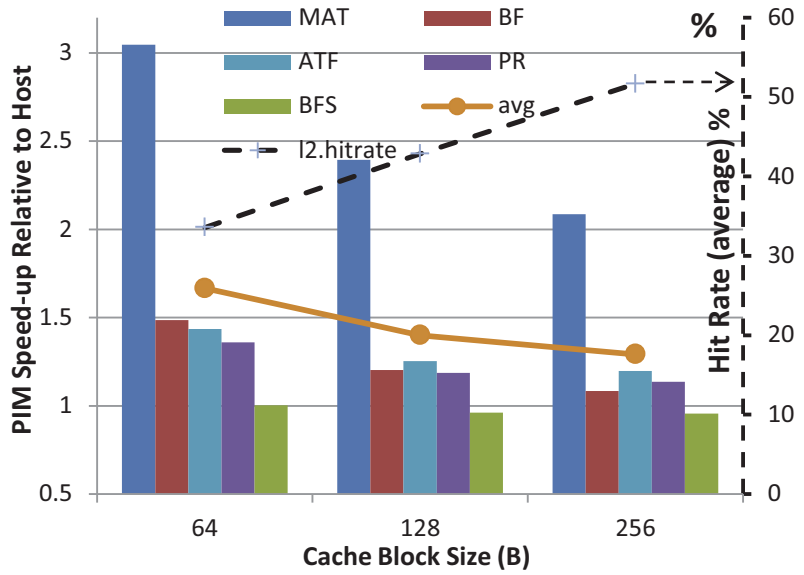


Figure 5.9: PIM's speed-up versus cache block size of the host

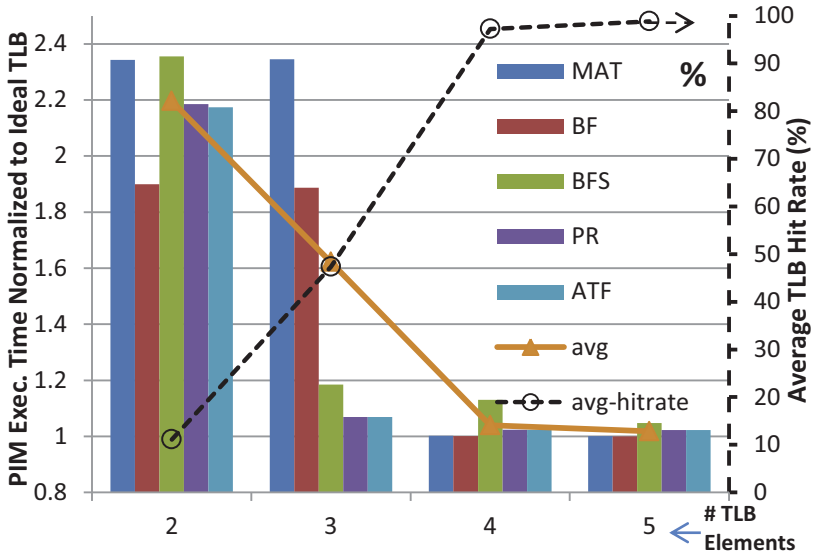


Figure 5.10: Effect of PIM's TLB size on hit-rate and speed-up

traversing of the graph nodes which results in a relatively high spatial locality (MAT in this plot represents simple 1000x1000 matrix addition as a reference for comparison). In our single-threaded experiments, the increase in the power consumption of the LLC cache was negligible, nevertheless in multi-programmed and parallel environments, the increased energy-per access for the increased cache block size can become critical.

Two additional experiment on PIM identify its optimal TLB size (Figure 5.10) and the required DMA transfer size (Figure 5.11). Four TLB elements were found enough for the studied computation kernels and a nearly perfect TLB hit-rate was

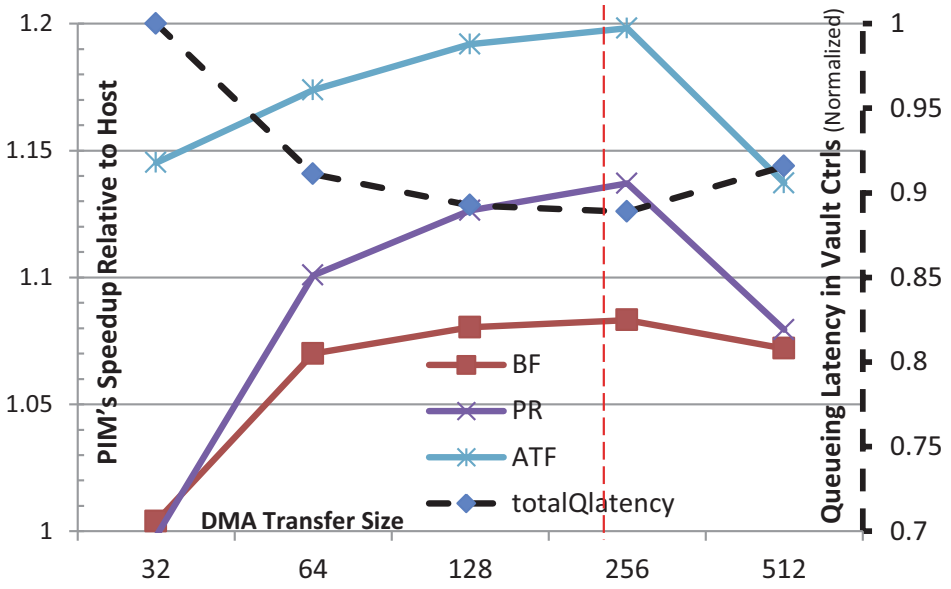


Figure 5.11: Effect of DMA transfer size on PIM's execution time

achieved. Also, transfer size for DMA Resource 1 in Figure 5.6 was changed from 32B to 512B and the optimal point was found around 256B transfers. This is because small transfers cannot hide memory access latency very well, and too large transfers incur larger queuing latency in the vault controllers, as the size of the row buffers is fixed at 256B.

We accounted for the energy consumption of each component type differently (assuming 28nm Low Power technology as the logic process). For the interconnects, energy/transaction was extracted from logic synthesis of the AXI-4.0 RTL model (See chapter 4) Cache power consumption was extracted from the latest release of CACTI [155] using low-power transistor models. Power consumed in the DRAM devices was extracted from DRAMPower [156], and verified against Micron's excel sheets. The energy consumed in the vault controllers was estimated to be 0.75pJ/bit [157]. SMC Controller was estimated to consume 10pJ/bit by scaling values obtained in [150]. For serial links, energy per transaction was considered 13.7pJ/bit [158], and an idle power consumption of 1.9 Watts (for transmission of the null flits) was estimated based on the maximum power reported in [91] and link efficiency in [54]. Also, since power state transition for the serial links introduces long sleep latency in the order of a few hundred nanoseconds, and a wakeup latency of a few microseconds [140], we assumed that during host's computations, links are fully active, while when PIM starts computing, links can be power gated [140]. For the processors, percentage of active/idle cycles were extracted from gem5, and the power consumption for each one were estimated based on

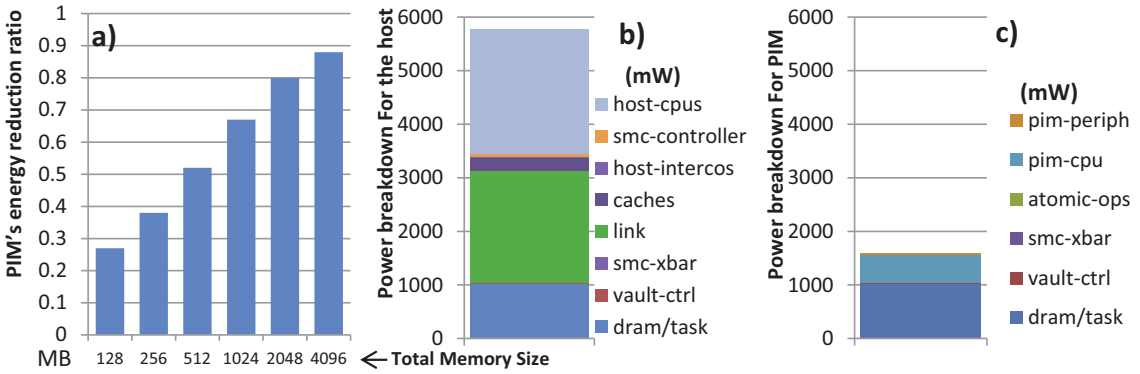


Figure 5.12: Achieved energy efficiency for using PIM versus total stacked memory size (a), power breakdown for execution of the host (b) and PIM (c)

[159][160]. Lastly, the energy consumptions associated with atomic commands, PIM's TLB, and its DMA engine were estimated based on logic synthesis and were found to be negligible.

To perform a fair analysis of the energy efficiency achieved by PIM, we omitted the system-background power and only considered the energy consumption related to the execution of tasks. Background power consists of all components which consume energy whether host is active or PIM, and can range from system's clocking circuits to peripheral devices, the secondary storage, the cooling mechanism, as well as, unused DRAM cells. One important observation was that for typical social graphs with less than 1 million nodes the total allocated DRAM size was always less than 100MB (Using LIL representation). While, a significant amount of power is consumed in the unused DRAM cells. In fact, increasing stacked DRAM's size is one of the background sources which can completely neutralize the energy efficiency achieved by PIM (See Figure 5.12.a). For this reason, we only consider the energy consumed in the "used" DRAM pages, all components in LoB of the memory cube, the serial links, the host processors, and their memory interface including the interconnects and the caches. Power consumption breakdown for execution of the task on PIM and the host are illustrated in Figure 5.12.b,c. The main contributors were found to be DRAM, the processors, and the serial links. The voltage and frequency of PIM's processor (on LoB) have been scaled down from 2GHz@1.05V to 1GHz@0.76V [159]. Under these circumstances, PIM can reduce power consumption by 3X.

To find the optimal point in terms of energy efficiency we scaled the voltage and frequency of PIM's processor and plotted PIM's performance per watts normalized to

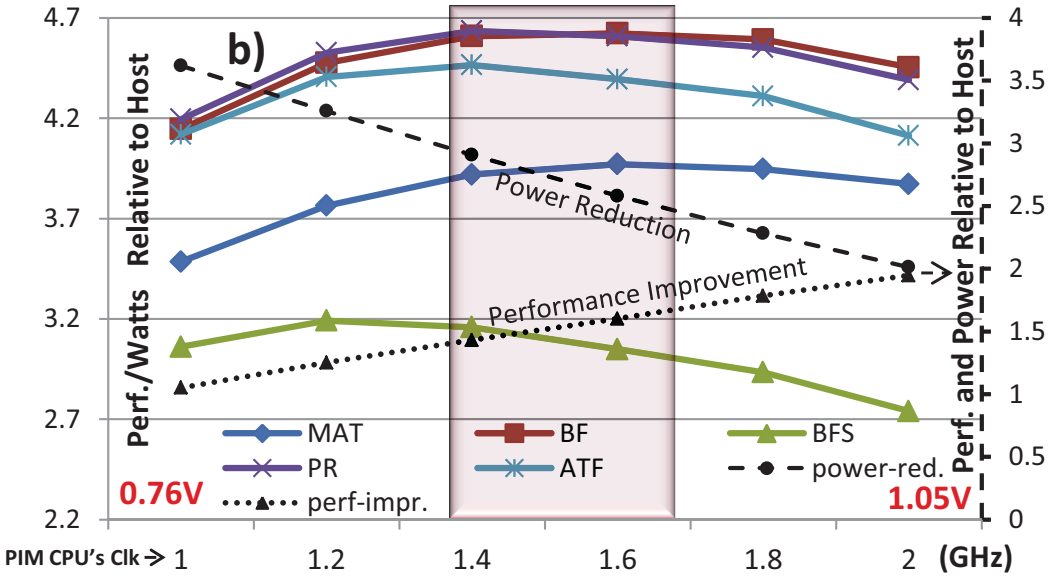


Figure 5.13: PIM's energy efficiency versus its clock frequency.

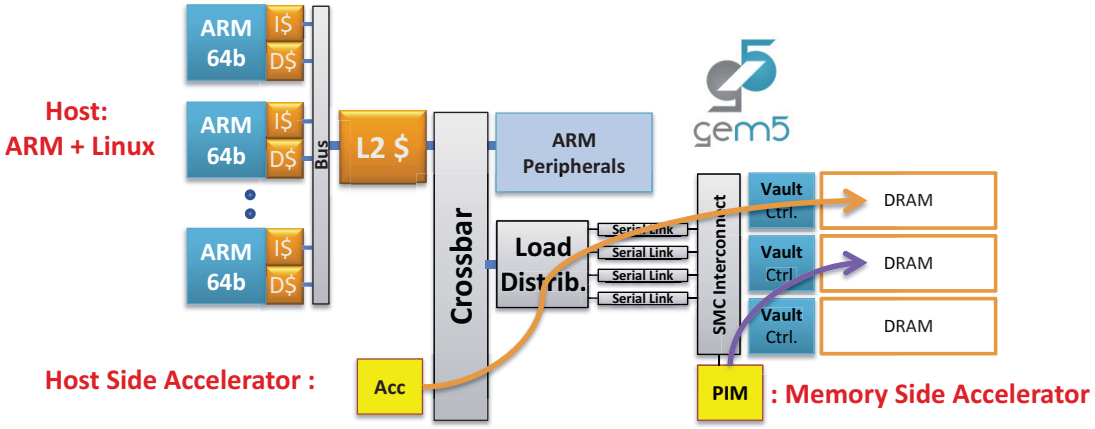


Figure 5.14: Simulation setup for comparison of PIM with a host-side accelerator with similar capabilities.

the host. As can be seen in Figure 5.13, clocking PIM at around 1.5GHz leads to highest energy efficiency in all cases.

As the final experiment, PIM was compared with a host-side accelerator with similar capabilities, to study the effect of vicinity to the main memory. For this purpose, we detached the complete PIM subsystem from the cube and connected it to "Interconnect 2" (Figure 5.1) without any change in its capabilities. An overview of the simulation setup for this experiment is shown in Figure 5.14.

For matrix addition, no performance difference was observed, because the DMA

engine effectively hides memory access latency. However, in all four graph traversal benchmarks PIM beats the host-side accelerator by a factor of 1.4X to 1.6X. This can be explained by the latency sensitivity of the graph traversal benchmarks (Average memory access latency from PIM to the main memory was measured as 46ns, while for the host side accelerator this value had increased to 74ns). Also, since the host side accelerator needs the serial links and the SMC Controller to be active, under the same conditions as the previous experiment (Considering power for the active banks of the DRAM and scaling down the voltage and frequency of PIM's processor), our PIM achieves an energy reduction of 55% compared to a similar accelerator located on the host. Lastly, according to Little's law [161], the host side accelerator requires more buffering to maintain the same bandwidth, due to higher memory access latency. This results in increased manufacturing cost and energy.

5.7 Summary

In this chapter we presented the first exploration steps towards design of SMC, a new PIM architecture that enhances the capabilities of the LoB die in HMC. An accurate simulation environment called SMCSim has been developed, along with a full featured software stack. A full system analysis of near memory computation has been performed including software to firmware and hardware layers, and considering offloading and dynamic overheads caused by the operating system, cache coherence, and memory management. A zero-copy pointer passing mechanism was devised to allow low overhead data sharing between the host and the PIM. Benchmarking results demonstrate that even in a case where the only benefit of using PIM is latency reduction, up to 2X performance improvement in comparison with the host SoC, and around 1.5X against a similar host-side accelerator is achievable. Moreover, by scaling down the voltage and frequency of the proposed PIM it is possible to reduce energy by about 70% and 55% in comparison with the host and the accelerator, respectively. A overview of the conclusions obtained throughout this thesis are presented in chapter 6, and ongoing and future directions are identified, as well.

Chapter 6

Conclusions

In this dissertation, we studied the effectiveness of the 3D integration technology and the optimization opportunities which it can provide in the different layers of the memory hierarchy in cluster-based many-core platforms.

In chapter 2 we looked inside a processing cluster and studied the applicability of 3D integration to shared L1 TCDMs. We presented two synthesizable 3D network architectures: C-LIN and D-LIN for shared L1 memory access, and demonstrated their effectiveness in comparison with conventional network on chips (NoC). In comparison with their 2D alternatives, our proposed designs offer modularity and better scalability. However, in terms of delay, they are so competitive with their 2D counterpart. This is due to the fact that the pipelines of the processors are extremely sensitive to the access latency of L1 memories, and current TSV technologies are not yet so competitive with on-chip wires. Therefore, small sized L1 memories are not beneficial in terms of performance to be moved towards the third dimension.

In chapter 3 we focused on out of the cluster L2 scratchpad memories because of their large required size and higher tolerance to latency and its variations. We presented a synthesizable 3D-stackable L2 memory IP component (3D-NUMA), to be attached to a cluster-based multi-core platform through its NIs and provide high-bandwidth L2 memory access with low average latency. 3D-NUMA implements a scalable non-uniform memory access (NUMA) architecture, allows stacking of multiple identical memory dies, supports multiple outstanding transactions, and achieves high clock frequencies due to its highly pipelined nature. We implemented 3D-NUMA with STMicroelectronics CMOS-28nm Low Power Technology and obtained a clock frequency of 500 MHz, limited by the access time of the memory arrays while its logic components could operate up to 1 GHz (up to 4 MB in 8 stacked dies with a memory density loss

of 9%). It was shown that addition of 3D-NUMA to a multi-cluster system can lead to an average performance boost of 34%. Further experiments and estimations confirmed that 3D-NUMA is energy and power efficient, temperature friendly, and has unique features suitable for low cost manufacturing.

In chapter 4 and chapter 5, we moved towards the last level in the memory hierarchy, and studied the benefits provided by 3D integration, there. We did not try to justify the effectiveness of 3D integration in this context because several academic and industrial examples are available with their most outstanding example being HMC. We focused on another important artifact of heterogeneous 3D integration, the possibility of near memory computation. We chose HMC as our target candidate and proposed the "Smart Memory Cube (SMC)", a fully backward compatible and modular extension to it, supporting near memory computation on its Logic Base (LoB). In chapter 4, we presented a high performance AXI-compatible interconnect to support this feature. Cycle accurate simulation demonstrated that, the proposed interconnect can easily meet the demands of current and future projections of HMC (Up to 205 GB/s READ bandwidth with 4 serial links and 32 memory vaults). The interference between the PIM traffic and the main links was found to be negligible when PIM has up to 2 ports requesting up to 64 GB/s. It was shown that low-interleaved addressing is not reliable enough for an abstracted memory such as HMC. Fat data structures with power-of-two node sizes were particularly identified as unfavorable patterns for low-interleaving. A more robust address scrambling mechanism was proposed and it was shown to effectively reduce bank/vault conflicts. Logic synthesis confirmed that our proposed models are implementable and effective in terms of power, area, timing (power consumption less than 5mW up to 1 GHz and area less than $0.4mm^2$).

Finally, in chapter 5 we presented the first exploration steps towards design of a PIM architecture for SMC. An accurate simulation environment called SMCSim was developed along with a full featured software stack. A full system analysis of near memory computation was performed including software to firmware and hardware layers, considering offloading and dynamic overheads caused by the operating system, cache coherence, and memory management. A zero-copy pointer passing mechanism was devised to allow low overhead data sharing between the host and the PIM. It was shown that even in a case where the only benefit of using PIM is latency reduction, up to 2X performance improvement in comparison with the host SoC, and around 1.5X against a similar host-side accelerator is achievable. By scaling down the voltage and frequency of the proposed PIM it is possible to reduce energy by about 70% and 55%

in comparison with the host and the accelerator, respectively.

As an ongoing work, we have focused on specific killer application domains where even more significant improvements are expected from PIM. We plan to extend SMC and the PIM architecture to a highly optimized parallel acceleration engine for those application domains. From the architectural point of view, we will strive to maintain flexibility and programmability. Thus, we will use a cluster of light-weight processors as a generic and low power infrastructure for acceleration of parallel applications. Also, by augmenting the vault controller and/or memory plane with computational capabilities (e.g. atomic operations), and connecting them using specific interconnects (e.g. ring networks), we will be able to accelerate the regular parallel kernels, even better.

References

[1] E. Azarkhish, I. Loi, and L. Benini, "A case for three-dimensional stacking of tightly coupled data memories over multi-core clusters using low-latency interconnects," *Computers Digital Techniques, IET*, vol. 7, no. 5, pp. 191–199, 2013.

[2] E. Azarkhish, I. Loi, and L. Benini, "A high-performance multiported L2 memory IP for scalable three-dimensional integration," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*, pp. 1–8, Oct 2013.

[3] E. Azarkhish, I. Loi, and L. Benini, "3D logarithmic interconnect: Stacking multiple L1 memory dies over multi-core clusters," in *Networks on Chip (NoCS), 2013 Seventh IEEE/ACM International Symposium on*, pp. 1–2, April 2013.

[4] E. Azarkhish, D. Rossi, *et al.*, "A modular shared L2 memory design for 3-D integration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, pp. 1485–1498, Aug 2015.

[5] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "A logic-base interconnect for supporting near memory computation in the hybrid memory cube," in *WoNDP: 2nd Workshop on Near-Data Processing In conjunction with the 47th IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*, 2014.

[6] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "High performance AXI-4.0 based interconnect for extensible smart memory cubes," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, (San Jose, CA, USA), pp. 1317–1322, EDA Consortium, 2015.

[7] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Design and evaluation of a processing-in-memory architecture for the smart memory cube," in *To appear in the Architecture of Computing Systems (ARCS)*, ARCS'16, 2016.

[8] P. M. Kogge and D. R. Resnick, "Yearly update: Exascale projections for 2013," Tech. Rep. SAND2013-9229, Sandia National Laboratoris, Oct. 2013.

[9] C. Santos, P. Vivet, D. Dutoit, P. Garrault, *et al.*, "System-level thermal modeling for 3d circuits: Characterization with a 65nm memory-on-logic circuit," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*, pp. 1–6, Oct 2013.

[10] "Micron sampling 2GB hybrid memory cube." http://techreport.com/news/25429/micron-sampling-2gb-hybrid-memory-cube, 2013.

[11] P. Ruch, T. Brunschwiler, W. Escher, S. Paredes, and B. Michel, "Toward five-dimensional scaling: How density improves efficiency in future computers," *IBM Journal of Research and Development*, vol. 55, pp. 15:1–15:13, Sept 2011.

[12] "Nvidia's fermi gp-gpu." http://www.pcper.com/reviews/Graphics-Cards/NVIDIA-Fermi-Next-Generation-GPU-Architecture-Overview, 2009.

[13] "Hybrid memory cube specification 2.1." Hybrid Memory Cube Consortium, 2015.

[14] S. Hong, S. Salihoglu, J. Widom, and K. Olukotun, "Simplifying scalable graph processing with a domain-specific language," in *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '14, (New York, NY, USA), pp. 208:208–208:218, ACM, 2014.

[15] M. Islam, M. Scrbak, K. Kavi, *et al.*, "Improving node-level MapReduce performance using processing-in-memory technologies," in *Euro-Par 2014: Parallel Processing Workshops*, vol. 8806 of *Lecture Notes in Computer Science*, pp. 425–437, Springer International Publishing, 2014.

[16] D. P. Zhang, N. Jayasena, A. Lyashevsky, *et al.*, "A new perspective on processing-in-memory architecture design," in *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, MSPC '13, (New York, NY, USA), pp. 7:1–7:3, ACM, 2013.

[17] D. Patterson, K. Asanovic, A. Brown, *et al.*, "Intelligent RAM (IRAM): the industrial setting, applications, and architectures," in *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, pp. 2–7, Oct 1997.

[18] F. Hamzaoglu, U. Arslan, N. Bisnik, *et al.*, "13.1 a 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 230–231, Feb 2014.

[19] A. Rahimi *et al.*, "A fully-synthesizable single-cycle interconnection network for shared-L1 processor clusters," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pp. 1–6, 2011.

[20] W. Davis *et al.*, "Demystifying 3D ICs: the pros and cons of going vertical," *Design Test of Computers, IEEE*, vol. 22, no. 6, pp. 498–510, 2005.

[21] S. Gupta *et al.*, "Techniques for producing 3-D ICs with high-density interconnect," in *Int. VLSI Multi-Level Interconnect. Conf.*, (Waikoloa Beach, HI, USA), 2004.

[22] G. Van der Plas *et al.*, "Design issues and considerations for low-cost 3D TSV IC technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 148–149, 2010.

[23] R. Radojcic, "Roadmap for design and EDA infrastructure for 3D products." http://www.eda.org/edps/EDP2012/Papers/3D Riko Radojcic Keynote.pdf, Apr. 2012.

[24] P. Vivet, D. Dutoit, Y. Thonnart, and F. Clermidy, "3D NoC using through silicon via: An asynchronous implementation," in *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*, pp. 232–237, 2011.

[25] F. Clermidy, D. Dutoit, E. Guthmuller, I. Miro-Panades, and P. Vivet, "3D stacking for multi-core architectures: From WIDEIO to distributed caches," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pp. 537–540, May 2013.

[26] D. U. Lee, K. W. Kim, K. W. Kim, *et al.*, "25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 432–433, Feb 2014.

[27] B. Giridhar, M. Cieslak, D. Duggal, *et al.*, "Exploring DRAM organizations for energy-efficient and resilient exascale memories," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, (New York, NY, USA), pp. 23:1–23:12, ACM, 2013.

[28] D. H. Woo, N. H. Seong, D. Lewis, and H.-H. Lee, "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1–12, 2010.

[29] A. Zia, P. Jacob, J.-W. Kim, *et al.*, "A 3D cache with ultra-wide data bus for 3D processor-memory integration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 967–977, 2010.

[30] J. Jung, K. Kang, and C.-M. Kyung, "Design and management of 3D-stacked NUCA cache for chip multiprocessors," in *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*, GLSVLSI '11, (New York, NY, USA), pp. 91–96, ACM, 2011.

[31] D. H. Woo, N. H. Seong, and H.-H. Lee, "Pragmatic integration of an SRAM row cache in heterogeneous 3D DRAM architecture using TSV," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 1–13, 2013.

[32] "Snapdragon 800 processors." http://www.qualcomm.com/.

[33] "Samsung exynos 5 octa processors." http://www.samsung.com/.

[34] Yole-Developpement, "3D IC & TSV interconnects 2012 business update." http://www.i-micronews.com/reports/3dic-tsv-interconnects-2012-business-update/8/302/, 2012.

[35] E. J. Vardaman, "3D IC with TSV: Status and developments." http://connection.ebscohost.com/c/articles/86024505/3d-ic-tsv-status-developments, Mar. 2013.

[36] A. Syed, "Emerging IC packaging technologies." http://www.smta.org/chapters/files/arizona-sonora_amkor _smta_az_expo_2012dec4.pdf, Dec. 2012.

[37] V. Solberg, S. McElrea, and W. Zohni, "New 3D packaging approach for next generation high performance DRAM," tech. rep., Invensas Corporation, San Jose, California USA, 2012.

[38] R. Crisp, "High performance DRAM packaging trends and solutions." http://www.tessera.com/abouttessera/upcomingevents/documents/dram-richard.pdf, Nov. 2010.

[39] A-Star-IME, "TSV silicon interposer for high I/O applications." http://www.ime.a-star.edu.sg/uploadfiles /3_proposal-tsv-interposer.pdf, Nov. 2010.

[40] S. Anderson, "Advanced packaging for mobile and growth products." http://www.smta.org/chapters/files/arizona-sonora_smta_tempe_7dec11_tech_forum_scl.pptx.pdf, Dec. 2011.

[41] Freescale-Semiconductor, "Freescale's redistributed chip packaging." http://www.freescale.com/files/shared/doc/reports_presentations/rcppresentation.pdf, Jan. 2013.

[42] "The next generation CUDA architecture, code named Fermi." White Paper, Sept. 2009.

[43] "The hypercore architecture." White Paper, Jan. 2010.

[44] D. Melpignano, L. Benini, E. Flamand, *et al.*, "Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications," in *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, (New York, NY, USA), pp. 1137–1142, ACM, 2012.

[45] D. Elliott, W. Snelgrove, and M. Stumm, "Computational RAM: A memory-SIMD hybrid and its application to DSP," in *Custom Integrated Circuits Conference, 1992., Proceedings of the IEEE 1992*, pp. 30.6.1–30.6.4, May 1992.

[46] D. Patterson, T. Anderson, N. Cardwell, *et al.*, "A case for intelligent RAM," *Micro, IEEE*, vol. 17, pp. 34–44, Mar 1997.

[47] R. Balasubramonian, J. Chang, T. Manning, *et al.*, "Near-data processing: Insights from a MICRO-46 workshop," *Micro, IEEE*, vol. 34, pp. 36–42, July 2014.

[48] G. H. Loh, N. Jayasena, M. Oskin, *et al.*, "A processing in memory taxonomy and a case for studying fixed-function pim," in *Workshop on Near-Data Processing (WoNDP)*, Dec 2013.

[49] A. Farmahini-Farahani, J. Ahn, K. Compton, and N. Kim, "DRAMA: An architecture for accelerated processing near memory," *Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2014.

[50] Z. Sura, A. Jacob, T. Chen, *et al.*, "Data access optimization in a processing-in-memory system," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, (New York, NY, USA), pp. 6:1–6:8, ACM, 2015.

[51] D. Zhang, N. Jayasena, A. Lyashevsky, *et al.*, "TOP-PIM: Throughput-oriented programmable processing in memory," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, (New York, NY, USA), pp. 85–98, ACM, 2014.

[52] G. Stelle, S. L. Olivier, D. Stark, A. F. Rodrigues, and K. S. Hemmert, "Using a complementary emulation-simulation co-design approach to assess application readiness for processing-in-memory systems," in *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, Co-HPC '14, (Piscataway, NJ, USA), pp. 64–71, IEEE Press, 2014.

[53] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, (New York, NY, USA), pp. 105–117, ACM, 2015.

[54] P. Rosenfeld, *Performance Exploration of the Hybrid Memory Cube*. PhD thesis. Univ. of Maryland, 2014.

[55] K. Kang, L. Benini, and G. Micheli, "A high-throughput and low-latency interconnection network for multi-core clusters with 3-D stacked L2 tightly-coupled data memory," in *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*, pp. 283–286, 2012.

[56] S. Borkar, "Networks for multi-core chips: A contrarian view," in *Symp. Low Power Electron. Design (ISLPED)*, 2007.

[57] R. A. Haring *et al.*, "The IBM Blue Gene/Q compute chip," *IEEE Micro*, vol. 32, no. 2, pp. 48–60, 2012.

[58] S. Satpathy *et al.*, "A 1.07 Tbit/s 128x128 swizzle network for SIMD processors," in *VLSI Circuits (VLSIC), 2010 IEEE Symposium on*, pp. 81–82, 2010.

[59] 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, (New York, NY, USA), pp. 187–198, ACM, 2006.

[60] R. Das *et al.*, "Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 175–186, 2009.

[61] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," *Computer Architecture Letters*, vol. 6, no. 2, pp. 37–40, 2007.

[62] K. Sewell *et al.*, "Swizzle-switch networks for many-core systems," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 2, no. 2, pp. 278–294, 2012.

[63] G. Beanato *et al.*, "3D-LIN: A configurable low-latency interconnect for multi-core clusters with 3D stacked L1 memory," in *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*, pp. 30–35, 2012.

[64] H. Saito, M. Nakajima, T. Okamoto, *et al.*, "A chip-stacked memory for on-chip SRAM-rich SoCs and processors," in *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pp. 60–61,61a, 2009.

[65] D. H. Kim *et al.*, "3D-MAPS: 3D massively parallel processor with stacked memory," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 188–190, 2012.

[66] K. Ito *et al.*, "Hierarchical 3D interconnection architecture with tightly-coupled processor-memory integration," in *3D Systems Integration Conference (3DIC), 2010 IEEE International*, pp. 1–6, 2010.

[67] E. Rosenbaum, V. Shukla, and M.-S. Keel, "ESD protection networks for 3D integrated circuits," in *3D Systems Integration Conference (3DIC), 2011 IEEE International*, pp. 1–7, 2012.

[68] V. Pavlidis, I. Savidis, and E. Friedman, "Clock distribution networks in 3D integrated systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 2256–2266, 2011.

[69] H. Xu, V. F. Pavlidis, and G. De Micheli, "Effect of process variations in 3D global clock distribution networks," *J. Emerg. Technol. Comput. Syst.*, vol. 8, pp. 20:1–20:25, Aug. 2012.

[70] I. Loi, F. Angiolini, and L. Benini, "Developing mesochronous synchronizers to enable 3D NoCs," in *Design, Automation and Test in Europe, 2008. DATE '08*, pp. 1414–1419, 2008.

[71] R. Dreslinski *et al.*, "Centip3De: A 64-core, 3D-stacked near-threshold system," *Micro, IEEE*, vol. 33, no. 2, pp. 8–16, 2013.

[72] A. Jain, "Research challenges and opportunities in 3D integrated circuits." www.usu.edu/mrc/Ankur_Jain.pdf, Jan. 2009.

[73] X. Dong and Y. Xie, "System-level cost analysis and design exploration for three-dimensional integrated circuits (3D ICs)," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pp. 234–241, 2009.

[74] E. J. Marinissen *et al.*, "Wafer probing on fine wafer probing on fine-pitch pitch micro bumps for 2.5d and 3D SICs." http://www.swtest.org/swtw_library/2011proc/PDF/ S04_03_Marinissen_SWTW2001.pdf, June 2011.

[75] S. Vangal *et al.*, "A 5.1GHz 0.34mm2 router for network-on-chip applications," in *VLSI Circuits, 2007 IEEE Symposium on*, pp. 42–43, 2007.

[76] L. Benini *et al.*, "MPARM: Exploring the multi-processor SoC design space with SystemC," *J. VLSI Signal Process. Syst.*, vol. 41, pp. 169–182, Sept. 2005.

[77] "SW/HW extensions for heterogenous multicore platforms (vIrtical)." http://www.virtical.eu/.

[78] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pp. 423–428, 2009.

[79] A. Kumar *et al.*, "A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS," in *ICCD*, pp. 63–70, 2007.

[80] D. Park *et al.*, "MIRA: A multi-layered on-chip interconnect router architecture," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 251–261, 2008.

[81] N. Khan, S. Alam, and S. Hassoun, "Power delivery design for 3D ICs using different through-silicon via (TSV) technologies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 4, pp. 647–658, 2011.

[82] G. L. Loi, B. Agrawal, N. Srivastava, *et al.*, "A thermally-aware performance analysis of vertically integrated (3D) processor-memory hierarchy," in *Design Automation Conference, 2006 43rd ACM/IEEE*, pp. 991–996, 2006.

[83] "Keystone II multicore DSP." http://www.ti.com/.

[84] J. Barth, D. Plass, E. Nelson, C. Hwang, *et al.*, "A 45nm SOI embedded DRAM macro for POWER7TM 32MB on-chip L3 cache," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 342–343, Feb 2010.

[85] E. Karl, Y. Wang, Y.-G. Ng, Z. Guo, *et al.*, "A 4.6GHz 162Mb SRAM design in 22nm tri-gate CMOS technology with integrated active VMIN-enhancing assist circuitry," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 230–232, Feb 2012.

[86] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (L3Cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 143–154, Feb 2013.

[87] J. Meng, K. Kawakami, and A. Coskun, "Optimizing energy efficiency of 3D multicore systems with stacked DRAM under power and thermal constraints," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pp. 648–655, 2012.

[88] M. Facchini, P. Marchal, F. Catthoor, *et al.*, "An RDL-configurable 3D memory tier to replace on-chip SRAM," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 291–294, 2010.

[89] G. Loh, "3D-stacked memory architectures for multi-core processors," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 453–464, 2008.

[90] G. Kumar, T. Bandyopadhyay, V. Sukumaran, *et al.*, "Ultra-high I/O density glass/silicon interposers for high bandwidth smart mobile applications," in *Electronic Components and Technology Conference (ECTC), 2011 IEEE 61st*, pp. 217–223, 2011.

[91] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*, pp. 87–88, June 2012.

[92] Q. Wu and T. Zhang, "Design techniques to facilitate processor power delivery in 3D processor-DRAM integrated systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 9, pp. 1655–1666, 2011.

[93] P.-J. Tzeng, J. Lau, C.-J. Zhan, *et al.*, "Process integration of 3D Si interposer with double-sided active chip attachments," in *Electronic Components and Technology Conference (ECTC), 2013 IEEE 63rd*, pp. 86–93, 2013.

[94] D. Secker, M. Ji, J. Wilson, *et al.*, "Co-design and optimization of a 256-GB/s 3D IC package with a controller and stacked DRAM," in *Electronic Components and Technology Conference (ECTC), 2012 IEEE 62nd*, pp. 857–864, 2012.

[95] J. H. Lau, "TSV interposer: The most cost-effective integrator for 3D IC integration," in *Chip Scale Review*, Sep 2011.

[96] "AMBA AXI protocol specification v2.0." ARM Holdings plc, 2010.

[97] C.-L. Lung, Y.-S. Su, H.-H. Huang, *et al.*, "Through-silicon via fault-tolerant clock networks for 3D ICs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 7, pp. 1100–1109, 2013.

[98] A.-C. Hsieh and T. Hwang, "TSV redundancy: Architecture and design issues in 3D IC," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, pp. 711–722, April 2012.

[99] D. H. Kim, K. Athikulwongse, and S. K. Lim, "Study of through-silicon-via impact on the 3D-stacked IC layout," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 862–874, 2013.

[100] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

[101] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[102] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, *et al.*, "A 1.2V 12.8GB/s 2Gb mobile Wide-I/O dram with 4x128 I/Os using TSV-based stacking," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pp. 496–498, 2011.

[103] X. Hu, P. Du, J. Buckwalter, and C.-K. Cheng, "Modeling and analysis of power distribution networks in 3D ICs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 2, pp. 354–366, 2013.

[104] N. Khan, S. Reda, and S. Hassoun, "Early estimation of TSV area for power delivery in 3D integrated circuits," in *3D Systems Integration Conference (3DIC), 2010 IEEE International*, pp. 1–6, 2010.

[105] A. Sridhar, A. Vincenzi, D. Atienza Alonso, and T. Brunschwiler, "3D-ICE: a compact thermal model for early-stage design of liquid-cooled ICs," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 1–4, 2013.

[106] A. Todri, S. Kundu, P. Girard, A. Bosio, L. Dilillo, and A. Virazel, "A study of tapered 3-D TSVs for power and thermal integrity," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, pp. 306–319, Feb 2013.

[107] D. Milojevic, H. Oprins, J. Ryckaert, *et al.*, "DRAM-on-logic stack - calibrated thermal and mechanical models integrated into pathfinding flow," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pp. 1–4, 2011.

[108] Y. K. Joshi and Y. J. Kim, "Thermal management of 3D-stacked IC." http://www.ipc.gatech.edu/materials/3d workshop joshi.pdf.

[109] Z. Feng and P. Li, "Fast thermal analysis on GPU for 3D ICs with integrated microchannel cooling," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 8, pp. 1526–1539, 2013.

[110] H. O. Yue Zhang and M. S. Bakir, "Within-tier cooling and thermal isolation technologies for heterogeneous 3D ICs," in *3DIC 2013*, 2013.

[111] "ARM Cortex-A12: The successor to the Cortex-A9 is available (in German)." http://www.elektroniknet.de/ halbleiter/prozessoren/artikel/100334/1/.

[112] "Internal communication with STMicroelectronics."

[113] J. Shah, "Estimating bond wire current-carrying capacity." https://www.idt.com/document/atc/power-systems-design-estimating-bond-wire-current-carrying-capacity, July 2012.

[114] B. Lee, L.-C. Wang, and M. Abadir, "Issues on test optimization with known good dies and known defective dies - a statistical perspective*," in *Test Conference, 2006. ITC '06. IEEE International*, pp. 1–10, 2006.

[115] U. Ahmed, G. Lemieux, and S. J. E. Wilton, "Performance and cost tradeoffs in metal-programmable structured ASICs (MPSAs)," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 2195–2208, 2011.

[116] N. Miyakawa, "A 3D prototyping chip based on a wafer-level stacking technology," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ASP-DAC '09, (Piscataway, NJ, USA), pp. 416–420, IEEE Press, 2009.

[117] A. Topol, D. La Tulipe, L. Shi, S. M. Alam, *et al.*, "Enabling SOI-based assembly technology for three-dimensional (3D) integrated circuits (ICs)," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pp. 352–355, Dec 2005.

[118] R. S. Mackay, H. Kamberian, and Y. Zhang, "Methods to reduce lithography costs with reticle engineering," *Microelectronic Engineering*, vol. 83, no. 49, pp. 914 – 918, 2006.

[119] D. Pramanik, H. H. Kamberian, C. J. Progler, *et al.*, "Cost effective strategies for ASIC masks," *Proc. SPIE*, vol. 5043, pp. 142–152, 2003.

[120] K. Jeong, A. B. Kahng, and C. J. Progler, "Cost-driven mask strategies considering parametric yield, defectivity, and production volume," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 10, no. 3, pp. 033021–033021–12, 2011.

[121] S. H. Pugsley, J. Jestes, R. Balasubramonian, *et al.*, "Comparing implementations of near-data computing with in-memory MapReduce workloads," *Micro, IEEE*, vol. 34, pp. 44–52, July 2014.

[122] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, (New York, NY, USA), pp. 336–348, ACM, 2015.

[123] Y. Kang, W. Huang, S.-M. Yoo, *et al.*, "FlexRAM: Toward an advanced intelligent memory system," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pp. 5–14, Sept 2012.

[124] Q. Zhu, B. Akin, H. Sumbul, *et al.*, "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3D Systems Integration Conference (3DIC), 2013 IEEE International*, pp. 1–7, Oct 2013.

[125] Y. Wang, Z. Shao, H. Chan, L. Bathen, and N. Dutt, "A reliability enhanced address mapping strategy for three-dimensional (3-D) NAND flash memory," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, pp. 2402–2410, Nov 2014.

[126] G. Wu, J. Gao, H. Zhang, and Y. Dong, "Improving PCM endurance with randomized address remapping in hybrid memory system," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pp. 503–507, Sept 2011.

[127] J. ling Yang, "Parallel interleavers through optimized memory address remapping," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, pp. 978–987, June 2010.

[128] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, pp. 2290–2302, Dec 2011.

[129] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 33, (New York, NY, USA), pp. 32–41, ACM, 2000.

[130] "Exploring memory management strategies in catamount." http://www.sandia.gov/ ktpedre/papers/mem-cug08.pdf, 2008.

[131] H. Vandierendonck and K. De Bosschere, "XOR-based hash functions," *Computers, IEEE Transactions on*, vol. 54, pp. 800–812, July 2005.

[132] C. Pfister, "Optimizing Memory Access Patterns for High-Performance Near-Memory Processing," Master's thesis, Swiss Federal Institute of Technology Zurich, 2015.

[133] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, pp. 216–231, Feb 2005.

[134] "Spectral density estimation." https://en.wikipedia.org/wiki/Spectral_density_estimation, 2015.

[135] "Substitution-permutation network." https://en.wikipedia.org/wiki/Substitution-permutation_network.

[136] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, *et al.*, "Present: An ultra-lightweight block cipher," in *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, (Berlin, Heidelberg), pp. 450–466, Springer-Verlag, 2007.

[137] "Double data rate (DDR) SDRAM." JEDEC JESD79F, 2005.

[138] G.Ramesh, V. Kumar, and K. Reddy, "Asynchronous FIFO design with gray code pointer for high speed AMBA AHB compliant memory controller," *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, vol. 1, pp. 32–37, Nov 2012.

[139] C. Weis, I. Loi, *et al.*, "An energy efficient DRAM subsystem for 3D integrated SoCs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pp. 1138–1141, March 2012.

[140] J. Ahn, S. Yoo, and K. Choi, "Low-power hybrid memory cubes with link power management and two-level prefetching," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[141] G. Kim, J. Kim, J. H. Ahn, and J. Kim, "Memory-centric system interconnect design with hybrid memory cubes," in *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pp. 145–155, Sept 2013.

[142] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*. Thompson Learning, 2008.

[143] H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-Btree: an update-conscious parallel directory structure," in *Data Engineering, 1999. Proceedings., 15th International Conference on*, pp. 448–457, Mar 1999.

[144] J. Zhong and B. He, "Towards GPU-accelerated large-scale graph processing in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, pp. 9–16, Dec 2013.

[145] F. Ji and X. Ma, "Using shared memory to accelerate MapReduce on graphics processing units," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 805–816, May 2011.

[146] R. Nair, "Evolution of memory architecture," *Proceedings of the IEEE*, vol. 103, pp. 1331–1345, Aug 2015.

[147] A. Hansson, N. Agarwal, A. Kolli, *et al.*, "Simulating DRAM controllers for future system architecture exploration," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pp. 201–210, March 2014.

[148] OpenSilicon, "HMC ASIC controller IP core." http://www.open-silicon.com/open-silicon-ips/hmc/.

[149] MALI, "Open source Mali-400/450 GPU kernel device drivers." http://malideveloper.arm.com/resources/drivers/.

[150] M. Schaffner, F. K. Gürkaynak, A. Smolic, and L. Benini, "DRAM or no-DRAM? exploring linear solver architectures for image domain warping in 28 nm CMOS," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, (San Jose, CA, USA), pp. 707–712, EDA Consortium, 2015.

[151] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, "Green-marl: A DSL for easy and efficient graph analysis," *SIGPLAN Not.*, vol. 47, pp. 349–362, Mar. 2012.

[152] G. Malewicz, M. H. Austern, A. J. Bik, *et al.*, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Con-*

ference on Management of Data, SIGMOD '10, (New York, NY, USA), pp. 135–146, ACM, 2010.

[153] S. Salihoglu and J. Widom, "GPS: A graph processing system," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, SSDBM, (New York, NY, USA), pp. 22:1–22:12, ACM, 2013.

[154] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data, June 2014.

[155] S. Wilton and N. Jouppi, "CACTI: an enhanced cache access and cycle time model," *Solid-State Circuits, IEEE Journal of*, vol. 31, pp. 677–688, May 1996.

[156] K. Chandrasekar, B. Akesson, and K. Goossens, "Improved power modeling of DDR SDRAMs," in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pp. 99–108, Aug 2011.

[157] B. Boroujerdian, B. Keller, and Y. Lee, "LPDDR2 memory controller design in a 28nm process."

[158] S. Lloyd and M. Gokhale, "In-memory data rearrangement for irregular, data-intensive computing," *Computer*, vol. 48, pp. 18–25, Aug 2015.

[159] J. Heinlen, "Leveraging advanced physical IP to deliver optimized SoC implementation at 40nm and below." Talk by ARM Physical IP Division, Nov. 2010.

[160] B. M. Tudor and Y. M. Teo, "On understanding the energy consumption of ARM-based multicore servers," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, pp. 267–278, June 2013.

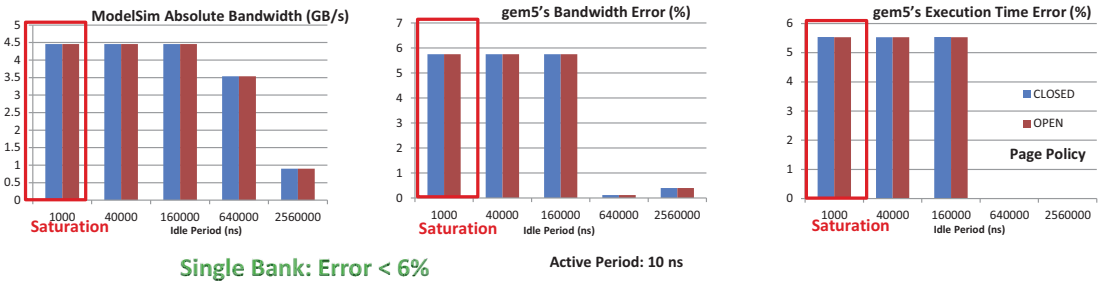
[161] "Little's law." https://en.wikipedia.org/wiki/Little's_law, 2015.

Appendix A

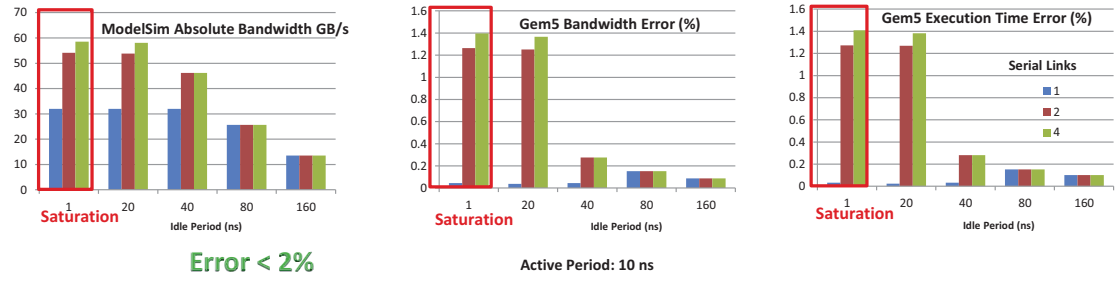
Accuracy verification of the high-level SMC model

This section presents a small subset of the accuracy comparison experiments performed between the high-level gem5 based SMC model developed in chapter 5, and the cycle-accurate model developed in chapter 4.

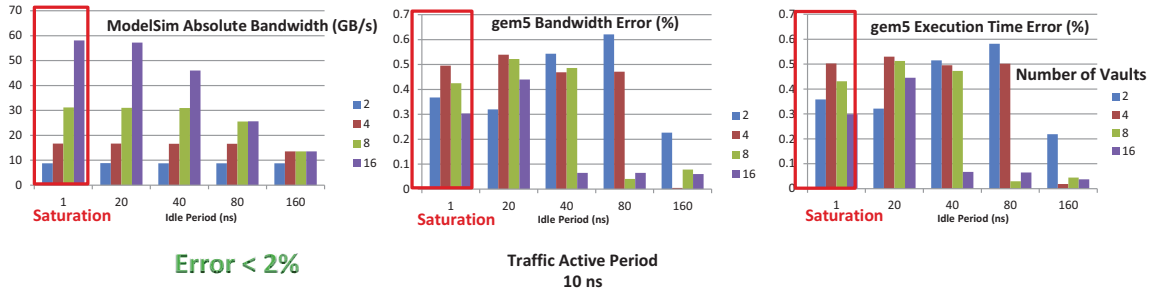
Comparison of a single memory bank between the two models: (1 serial link, 1 vault, 1 bank only)



Comparison of full HMC devices modeled in both environments, where the number of serial links has been altered:



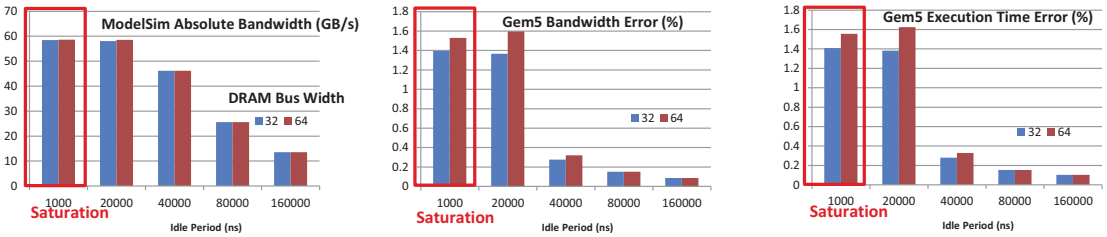
Comparison of full HMC devices modeled in both environments, where the number of memory vaults has been altered:



Error < 2%

Traffic Active Period
10 ns

Comparison of full HMC devices modeled in both environments, where DRAM bus width has been altered:



Error < 2%

Active Period: 10 ns

Appendix B

Source codes of computation kernels

Unoptimized source codes of the ATF and PR kernels:

a) Average Teenage Follower (ATF)

```
for ( ulong_t r=0; r<NODES; r++ )
    {
      if ( nodes[r].teenager )
        for ( ulong_t c=0; c<nodes[r].out_degree ; c++ )
        {
          node*succ = nodes[r].successors[c];
          succ->followers++;
        }
    }
```

b) Pagerank

```
for ( ulong_t i=0; i<NODES; i++ )
    {
      nodes[i].page_rank = 1.0 / NODES;
      nodes[i].next_rank = 0.15 / NODES;
    }
    ulong_t count = 0;
    float diff = 0.0;
    do {
      for ( ulong_t i=0; i<NODES; i++ )
      {
        float delta = 0.85 * nodes[i].page_rank / nodes[i].out_degree;
        for ( ulong_t j=0; j<nodes[i].out_degree; j++ ) // for node.successors
            nodes[i].successors[j]->next_rank += delta;
      }
      diff = 0.0;
      for ( ulong_t i=0; i<NODES; i++ )
      {
        diff += fabsf(nodes[i].next_rank - nodes[i].page_rank);
        nodes[i].page_rank = nodes[i].next_rank;
        nodes[i].next_rank = 0.15 / NODES;
      }
    } while (++count < PAGERANK_MAX_ITERATIONS && diff > PAGERANK_MAX_ERROR);
```

Unoptimized source codes of the BFS and BF kernels:

```
a)  Breadth First Search (BFS)
 ulong_t total_distance = 0;
     while ( !(queue_empty) )
     {
        ulong_t v = queue_top;
        queue_pop;
        for ( ulong_t c=0; c<nodes[v].out_degree ; c++ )
        {
           node*succ = nodes[v].successors[c];
           if (succ->distance == NC) // Infinite
           {
              succ->distance = nodes[v].distance + 1;
              total_distance += succ->distance;
              queue_push(succ->ID);
           }
        }
     }
b) Bellman-Ford
 ulong_t total_distance = 0;
     for ( unsigned r=0; r<NODES; r++ )
        for ( ulong_t c=0; c<nodes[r].out_degree; c++ ) // for node.successors
        {
           node*u = &nodes[r];
           node*v = nodes[r].successors[c];
           ulong_t w = nodes[r].weights[c];
           if ( u->distance != NC && v->distance > u->distance + w )
              v->distance = u->distance + w;
        }
```

DMA optimized version of matrix addition:

```
ping = &PIM_VREG[0];
pong = &PIM_VREG[0] + XFER_SIZE*3;

DMA_REQUEST(A, A_ping, XFER_SIZE, PIM_DMA_READ, DMA_RES0 );
DMA_REQUEST(B, B_ping, XFER_SIZE, PIM_DMA_READ, DMA_RES1 );
while(PIM_DMA_STATUS & DMA_RES0);
while(PIM_DMA_STATUS & DMA_RES1);

num_bursts = SS*sizeof(ulong_t)/XFER_SIZE;
for (x=0; x<num_bursts; x++)
{
   // Fill Pong (Request)
   if ( x+1 < num_bursts ) // Boundary
   {
      DMA_REQUEST(A+(x+1)*XFER_SIZE, A_pong, XFER_SIZE, PIM_DMA_READ, DMA_RES2 );
      DMA_REQUEST(B+(x+1)*XFER_SIZE, B_pong, XFER_SIZE, PIM_DMA_READ, DMA_RES3 );
   }

   // Work on Ping's data
   for ( j=0; j< XFER_SIZE/sizeof(ulong_t); j++ )
      ((ulong_t*)C_ping)[j] = ((ulong_t*)A_ping)[j] + ((ulong_t*)B_ping)[j];

   // Write back the result of Ping
   while(PIM_DMA_STATUS & DMA_RES4);
   DMA_REQUEST(C+x*XFER_SIZE, C_ping, XFER_SIZE, PIM_DMA_WRITE, DMA_RES4 );

   // Wait for Pong to finish
   while(PIM_DMA_STATUS & DMA_RES2);
   while(PIM_DMA_STATUS & DMA_RES3);

   // Swap ping and pong
   swap = ping;
   ping = pong;
   pong = swap;
}
while(PIM_DMA_STATUS & DMA_RES4);
```

Optimized ATF source code using two DMAs and an atomic HMC command:

```
DMA_REQUEST(nodes, nodes_pong, nodes_chunk, PIM_DMA_READ, DMA_RES0 );
  total_followers = 0;
  rr = nodes_count;
  num_pongs = -1;
  for ( r=0; r<NODES; r++ )
  {
    if ( rr == nodes_count )
    {
      rr = 0;
      DMA_WAIT( DMA_RES0);
      nodes_swap = nodes_ping; nodes_ping = nodes_pong; nodes_pong = nodes_swap; // Swap Ping and Pong
      DMA_REQUEST(&nodes[r+nodes_count], nodes_pong, nodes_chunk, PIM_DMA_READ, DMA_RES0 );
    }
    if ( nodes_ping[rr].teenager && nodes_ping[rr].out_degree )
      DMA_REQUEST(nodes_ping[rr].successors, succ_pong, nodes_ping[rr].out_degree *
        sizeof(node*), PIM_DMA_READ, DMA_RES1 );
    if ( num_pongs != -1 )
    {
      for ( c=0; c< num_pongs; c++ )
      {
        #ifdef USE_HMC_ATOMIC_CMD
        HMC_ATOMIC___INCR(succ_ping[c]->followers);
        #else
        succ_ping[c]->followers++;
        #endif
      }
      num_pongs = -1;
    }

    if ( nodes_ping[rr].teenager && nodes_ping[rr].out_degree )
    {
      num_pongs = nodes_ping[rr].out_degree;
      DMA_WAIT( DMA_RES1);
      succ_swap = succ_ping; succ_ping = succ_pong; succ_pong = succ_swap;
    }
    rr++;
  }
  // Termination
  if ( num_pongs != -1 )
  {
    for ( c=0; c< num_pongs; c++ )
    {
      #ifdef USE_HMC_ATOMIC_CMD
      HMC_ATOMIC___INCR(succ_ping[c]->followers);
      #else
      succ_ping[c]->followers++;
      #endif
    }
    num_pongs = -1;
  }
```