3D Stacked Memories for Digital Signal Processors

By

Daniel W. Chang

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONIN-MADISON

2013

Date of final oral examination: 07/25/2013

The dissertation is approved by the following members of the Final Oral Committee:

Mark D. Hill, Gene M. Amdahl Professor, Computer Sciences

Nam Sung Kim, Associate Professor, Electrical and Computer Engineering

Mikko H. Lipasti, Philip Dunham Reed Professor, Electrical and Computer Engineering

Katherine Morrow, Associate Professor, Electrical and Computer Engineering

Michael J. Schulte, Adjunct Associate Professor, Electrical and Computer Engineering

Acknowledgements

There is a long list of people who have contributed, either directly or indirectly, towards this dissertation. Without these people, I could not have succeeded at the University of Wisconsin. First and foremost among those I would like to thank are my parents, Dr. Chun Chang and Myeongheiu Chang for their unwavering support and encouragement throughout my life. They encouraged me at an early age to pursue a career in science and it was through their support (both financial and emotional) that I was able to receive my Electrical Engineering undergraduate degree at the University of Illinois. While unhappy with my position at Advanced Micro Devices (AMD), they also encouraged me to apply to Graduate School and "figure out what I wanted to do with my life" because I was still young. They could have just as easily talked me into staying with a stable, well-paying job, but they were right and now at the end of my graduate career I can finally say I have found something I am passionate about and that is teaching undergraduates.

I want to express my deepest gratitude to my advisers Dr. Michael Schulte and Dr. Nam Sung Kim for their support throughout my graduate studies. Their knowledge, work ethic, and guidance have been invaluable to me and I could not have done this without them. In particular, both Dr. Schulte and Dr. Kim have very busy schedules and yet both continued to make time for me whenever needed. I cannot express how appreciative I am of their efforts and I look forward to future collaborations after I start my new job as an Assistant Professor at Rose-Hulman Institute of Technology.

I would also like to thank Dr. Katherine Morrow and Dr. Mikko Lipasti. Throughout my graduate studies, I had the opportunity to teach under both professors a number of times and it was through their mentorship that I ultimately decided I wanted to become a professor at an

undergraduate, teaching-focused University. In fact, I feel Professor Lipasti gave me a gentle nudge towards this career when he nominated me for a teaching award two years ago. Both Professors were also extremely helpful during my faculty job search and without their mentorship and friendship; I would not be where I am today.

I would also like to thank Dr. Mark Hill for being a member of my dissertation committee. He made time to meet with me in private and give me feedback and that feedback has helped improve this dissertation considerably.

Lastly, I would like to thank the current and former colleagues in lab for their company and support. Specifically Amin Farmahini-Farahani, Andy Nere, Arsalan Zulfiqar, Chris Jenkins, Dan Seemuth, Dave Palframan, Felix Loh, Hamid Ghasemi, Hao Wang, Hsiang-Kuo Tang, Jake Adriaens, Jungseob Lee, Kyle Rupnow, Mitch Hayenga, Paula Aguilera, Philip Garcia, Sean Franey, Syed Gilani, Tony Gregerson, and Vignyan Reddy. Our discussions were not always technical, but their company made the graduate school journey much more enjoyable.

Abstract

Recently, three-dimensional (3D) integration technology has enabled researchers and engineers to explore novel architectures. Due to the growing memory requirements of modern signal processing applications, such as multimedia processing, speech recognition, and face recognition, it was thought that digital signal processors (DSPs) could greatly benefit from 3D memory integration technology where high-density memories are placed below processing cores. Until recently, it was believed that this integration could lower main memory latencies by 45% to 60%, which would reduce the average memory access time for these modern signal processing applications and improve performance while reducing energy consumption. Additionally, 3D memory integration technology also allowed a large increase in the main memory bus width by using small through silicon vias (TSVs) instead of off-chip metal wires. This large increase in the main memory bus width meant each main memory request could bring more data into the last-level on-chip memory and improve the performance of streaming signal processing applications whose memory access behavior exhibits a large amount of spatial locality.

My dissertation first provides a more accurate 3D main memory model that demonstrates that the latency reduction of going from conventional DDR2 DRAM to 3D memory technology (using the 65 nm technology node) is roughly 4% instead of the often quoted 45% to 60%. With this memory model, I re-evaluate the performance impact of 3D main memory on DSPs and find the performance benefits from the latency savings are very small.

I next analyze current 3D main memory with Wide I/O, which can lower main memory latencies by 15.9% and greatly increase the main memory bus width. I demonstrate that the latency improvements of 3D main memory with Wide I/O and increasing the main memory bus width from 64 bits to 4,096 bits can improve the average performance of key signal processing

applications by 9.7% (and up to 23.3%), but also increases average energy consumption by 2.6% (and up to 8.9%). For energy-constraint DSPs that are often found in mobile devices such as cellphones, this energy increase may be unacceptable.

To help mitigate this energy increase, my dissertation research proposes novel techniques to dynamically scale the effective main memory bus width of a DSP based on the program phases of an application. These bandwidth scaling algorithms increase the main memory bus width during memory intense program phases to improve performance and lower the bus width during compute intensive phases to improve I/O energy efficiency. These algorithms can improve average DSP performance by 6.6% (and up to 12.8%) while increasing average energy consumption by only 0.5%.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Dissertation Motivation

Digital signal processors (DSPs) are playing an increasingly important role in modern computing systems ranging from consumer electronics to health care and security systems [1]. Over the past two decades, DSPs have become one of the most important drivers of the global semiconductor industry, with nearly 1.6 billion DSPs shipped in 2008. In 2013, it is expected that over 3.3 billion DSPs will be shipped, which represents an average growth rate of 16% per year [2]. The global DSP market is estimated to grow from $6.2 billion in 2011 to $9.6 billion in 2016 [3]. With significant growth in portable electronics such as smartphones, embedded DSPs have become a major area of design and research.

Modern signal processing applications such as multimedia processing and speech recognition are becoming more compute and memory intense. However, delivering increased performance to meet these applications' demands is becoming more difficult. In response to the memory-intensive nature of signal processing applications, architects have been integrating more on-chip memory into DSPs, which traditionally have small on-chip memories. However, these solutions have not fully addressed the "memory wall" problem [4] for DSPs, in which the relative memory access latencies have been increasing with each new process generation.

Recently, researchers have begun exploring three-dimensional (3D) integration technology, in which logic and memory dies are vertically stacked together and connected using die-to-die or through-silicon vias (TSVs) [5]. Since each 3D layer can utilize a different process technology [6], 3D integration allows unique processor designs, including designs in which high-density dynamic random access memory (DRAM) is stacked below DSP cores. By stacking main

memory below the processor in the same package, it was believed that main memory access latencies could be much shorter than traditional out-of-package memory latencies, which would help directly address the memory wall problem. Furthermore, stacked DRAM can also lead to higher main memory bandwidth by using a large number of TSVs, which can increase the performance of applications that have high spatial locality [7].

Although 3D integration technology has been studied before, the vast majority of work focuses on exploring the impact of 3D technology in high-performance, general-purpose microprocessors and little research have been done in the field of DSPs. In particular, DSPs are commonly found on mobile devices that have stringent energy requirements, which require designers to strongly consider energy consumption with any design decision. My dissertation research looks to expand 3D integration technology research to include DSPs while also considering energy consumption.

1.2 Dissertation Contributions

My dissertation research aims to directly address the growing memory requirements of modern signal processing applications through the integration of DSPs and 3D DRAM with Wide I/O. Although DSPs seem like excellent candidates for 3D DRAM due to their small on-chip memories, my work demonstrates that the latency claims of 3D main memory need re-evaluation and simply stacking memory dies below a DSP does not lead to a large enough latency reduction to significantly improve the performance of signal processing applications. I first re-evaluate the latency claims of 3D DRAM and find the often quoted 45% to 60% latency reduction to be incorrect. I conclude that the key performance advantage of 3D DRAM lies in the capability to significantly increase the main memory bus width with a large number of TSVs.

I also demonstrate that increasing the main memory bus width with TSVs and using the additional bandwidth to fetch adjacent L2 cache lines can lead to considerable performance improvements for streaming multimedia applications that exhibit a high degree of spatial locality. However, this performance improvement comes at the cost of higher DSP energy consumption and simply increasing the main memory bus width with 3D technology can sometimes lead to an inefficient usage of the wider bus. My research shows that signal processing applications have program phases where the wider bus does not benefit the application and simply consumes more I/O energy. Since DSPs are commonly found in energy-constraint mobile devices, this energy increase may be unacceptable.

To address the increase in energy consumption that comes with increasing the main memory bus width I propose novel dynamic bandwidth scaling algorithms that increase or decrease the main memory bus width based on the program phases of the application. My bandwidth scaling algorithms maintain most of the performance improvement of increasing main memory bandwidth while also addressing the increase in energy consumption. These bandwidth scaling algorithms can achieve the majority of the performance improvements from increasing main memory bandwidth, but only spend 25.2% of the execution time in the maximum bus width setting, leading to better I/O energy efficiency. My proposed bandwidth scaling algorithms lead to a reduction in DSP energy consumption while still addressing the growing memory requirements of modern signal processing applications.

The key contributions of this dissertation include:

1. The presentation of a more accurate 3D DRAM latency model that shows the often quoted 45% to 60% latency savings is inaccurate and that simply stacking memory dies below the processor will decrease DRAM latency by only 4.1% [8].

2. A reevaluation of the performance benefits of 3D DRAM on DSPs with the updated latency model through cycle-accurate simulations. In these simulations, I find the performance benefits of 3D DRAM (without Wide I/O) are very small and that the key advantage of 3D DRAM is the ability to significantly increase the main memory bus width with additional TSVs [8] [9].
3. The presentation of a model for 3D DRAM with Wide I/O that not only significantly increases the main memory bus width, but also decreases main memory latency with two memory architecture changes [10].
4. The demonstration, through cycle-accurate simulations, that 3D DRAM with Wide I/O and adjacent cache line prefetching can increase the performance of modern signal processing applications by an average of 9.7% (and up to 23.3%), but this performance increase comes with an increase in average DSP energy consumption (2.6%) [10].
5. The proposal of dynamic bandwidth scaling algorithms, which increase or decrease the main memory bus width based on the phase behavior of the application [10]. To the best of my knowledge, these dynamic bandwidth scaling algorithms are the first of their kind.
6. The demonstration, through cycle-accurate simulations, that my dynamic bandwidth scaling algorithms maintain the majority of performance from increasing the main memory bus width while also reducing the increase in DSP energy consumption through better I/O efficiency [10].

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows: Chapter 2 provides background information on 3D integration technology, modern DSP architectures, and main memory. Chapter 3 provides a survey of related work. Chapter 4 presents the experimental framework

used throughout this dissertation including an explanation of the simulation environment, a description of the baseline processor, and a summary of the benchmarks used for evaluation and comparison. Chapter 5 presents an accurate 3D DRAM model. Chapter 5 also re-evaluates the performance impact of 3D DRAM with this more accurate memory model for a modern DSP. Chapter 6 presents a model for 3D DRAM with Wide I/O. In this chapter, 3D DRAM with Wide I/O is also evaluated on a modern DSP and results for performance and energy are analyzed. Chapter 7 proposes dynamic bandwidth scaling and evaluates their performance and energy impact for a modern DSP. Chapter 8 summarizes my research and Chapter 9 describes some opportunities for futures research in this area.

2 Background

This chapter provides background information on 3D integration technology including a short description of the three types of 3D technology. This chapter also includes background information on digital signal processors (DSPs) including how they differ from general-purpose processors, the types of applications they commonly run, and a description of the Texas Instrument Very Long Instruction Word (VLIW) C67x DSP that I used for this dissertation research. Lastly, this chapter gives a detailed explanation of main memory organization and an analysis of the steps involved in a main memory request.

2.1 3D Integration Technology

Three-dimensional (3D) integration technology is an emerging fabrication technology in which multiple integrated chips are vertically stacked. 3D integration technology offers a number of attractive features over traditional 2D integration technology, such as increased device density, greater routing flexibility and reduced wire lengths. A number of 3D integration technologies are currently being pursued and they can be divided into three categories:

1. 3D packaging technology is a die-to-die technology that uses wire bonding for vertical interconnections [11]. 3D packaging technology is already used in many commercial products, such as cell phones, but has very low inter-die interconnect density compared to other 3D technologies, making it less attractive for memory-intensive applications.
2. Transistor build-up 3D technology is a transistor-level integration technology that forms transistors inside on-chip interconnect layers [12], on poly-silicon films [13] or on single-crystal silicon films [14]. Although a high vertical interconnect density can be achieved,

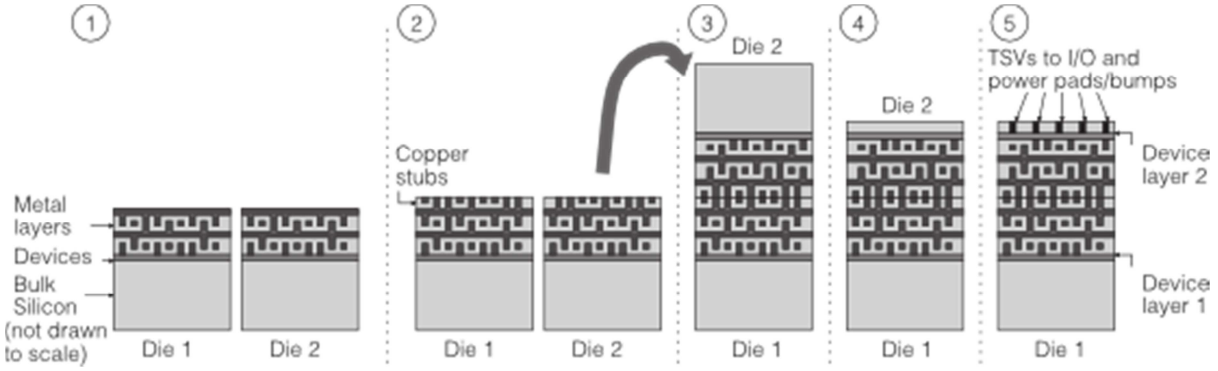

Figure 2.1: Fabrication steps for wafer-level, back-end-of-the-line 3D technology (from [6]).

it is not compatible with existing fabrication processes. It is also subject to severe process temperature constraints that can dramatically degrade the circuit's electrical performance.

3. Monolithic, wafer-level, back-end-of-the-line (BEOL) compatible 3D technology is a wafer-to-wafer technology that uses through-silicon vias (TSVs) to form inter-die interconnections [15]. TSVs have the potential to offer the greatest interconnect density, but also have a high cost. Figure 2.1 (taken from [6] with permission from Gabriel Loh) illustrates the fabrication steps for this type of 3D integration technology, which are described below:
    1. Two wafers are separately fabricated and processed using standard fabrication techniques.
    2. To assist with the fusing of the two wafers, copper is deposited into the top metal layer. This is similar to building conventional vias between metal layers.
    3. The two wafers are arranged face-to-face and subjected to thermal compression. The heat and pressure fuses the two copper deposits together thereby fusing the two dies.

4. Chemical-mechanical polishing (CMP) is used to thin one layer of the 3D stack to only 10 to 20 μm.
5. The thinning allows the through-silicon vias (TSVs) that implement the external I/O and power and ground connections to be relatively short; this minimize voltage losses.

For my dissertation research, I assume wafer-level BEOL-compatible 3D integration technology, since it appears to be the most promising option for high-volume production and offers a number of advantages including the ability to integrate different technologies (like CMOS logic and DRAM memories) and provide a massive amount of inter-die bandwidth via the TSVs. These two abilities allow performance-improving features, such as 3D main memory and larger capacity caches, to be implemented efficiently while also allowing designers to realize radically different system organizations.

2.2 Digital Signal Processors (DSP)

Digital signal processors (DSPs) are a type of specialized microprocessor optimized for signal processing applications. Many DSPs use a Very Long Instruction Word (VLIW) format to increase parallelism, and they are typically statically-scheduled. These features allow DSPs to reduce the hardware complexity of their design and have lower power consumption compared to general-purpose processors. This reduction in power consumption makes DSPs an attractive choice for embedded devices, which is why they are commonly found in popular smartphones and other multimedia devices. Digital signal processors differ from general-purpose microprocessors in a number of ways [16] [17] [18] [19]:

1. Many signal processing applications are deadline-driven and require the processor to complete the operations within some fixed amount of time.

2. Some DSP memory architectures are designed for streaming data and make use of DMA and software-controlled memories, such as scratchpad memories.
3. Although some DSPs use scratchpad memories, many still have traditional cache hierarchies. However, these caches are typically much smaller than those used in general-purpose processors.
4. Some DSPs use Very Long Instruction Word (VLIW) techniques for increased performance, lower power and deterministic program behavior.
5. Although general-purpose microprocessors can execute signal processing applications, they are not well suited for use in devices such as mobile phones due to energy and thermal constraints. DSPs are more specialized for these types of devices.

Since DSPs are commonly found in products such as portable electronics and multimedia devices, they often implement streaming applications, such as multimedia processing and wireless communications. These wireless communication applications are usually deadline-driven and may require the DSP to finish a particular code segment within a specified time frame. However, multimedia applications (unlike wireless communication applications) have soft deadlines, for which the application can tolerate a missed deadline and may respond by decreasing the quality of service. For my dissertation research, all the applications evaluated are multimedia signal processing applications that fit this behavior.

To address deadline-driven, streaming applications, many DSPs make use of software-controlled memories, such as scratchpad memories [20]. Software-controlled memories refer to a type of memory in which the programmer has knowledge of the memory hierarchy and internal latencies and programs the application with these in mind (although some of this can be handled by modern compilers). Since these memories allow for deterministic program execution times,

they help ensure that applications meet their deadlines. Furthermore, scratchpad memories also provide an energy savings since these memories do not need to perform tag comparisons or possibly look for data in multiple locations. 3D memory integration technology offers DSPs the ability to either integrate software-controlled scratchpad memories or easily increase the size of such memories. For my dissertation research, I chose not to look at software-controlled scratchpad memories since the DSPs found in mobile computing devices such as smart phones usually have cache memories.

However, although many DSPs use software-controlled scratchpad memories, a large number of DSPs (such as the Texas Instrument series of DSPs) still use memory hierarchies with L1 and L2 caches, similar to those found in general-purpose processors. However unlike general-purpose processors, DSPs often have significantly smaller cache sizes since the devices DSPs are traditionally found in have much smaller area constraints. It is common for DSPs to have L1 cache sizes as small as 4 KB. Although caches have probabilistic behavior, which does not address the deterministic requirement of many signal processing applications, they provide the advantages of a simpler programming model while also allowing flexibility when porting an application from one processor to another. With software-controlled memories, programmers may need to rewrite their programs for each new processor memory hierarchy.

3D memory integration technology offers the opportunity to integrate main memory in the same package as the processor and was believed to significantly decrease the memory latency, which decreases the cache miss penalty and decreases energy consumption. As will be discussed later in this document, we found the latency reduction claims for 3D memory to be overly aggressive and the decrease in memory latency from 3D memory to be significantly less than the often-quoted reductions.

Another architectural feature many DSP architectures take advantage of is using Very Long Instruction Word (VLIW) techniques to increase parallelism. VLIW architectures exploit instruction-level parallelism by allowing programmers or compilers to find independent instructions, dispatch them in parallel and allow them to execute concurrently on multiple execution units. Since many multimedia applications exhibit instruction-level parallelism and data-level parallelism that can be transformed into instruction-level parallelism, VLIW architectures take advantage of this to increase performance while keeping area and energy low by avoiding complex hardware structures for register renaming, bypass logic, and branch prediction found in many out-of-order superscalar processors. In addition, VLIW architectures also address the deterministic execution time requirements of many signal processing applications, since the programmer or compiler statically schedules instructions.

Lastly, although general-purpose processors can execute many signal processing applications, they generally do not perform as well as DSPs on these applications and consume too much energy and area to be effectively used in the types of devices that employ DSPs. In contrast, DSPs can have specialized instruction set architectures for signal processing applications and include hardware features such as an integer multiply accumulate (MAC). DSPs implement these applications at high performance with low energy and area, thus making them better candidate architectures for portable electronics and multimedia devices.

For my dissertation research, the baseline DSP chosen for evaluation is the Texas Instrument C67x DSP [21]. The C67x is an eight-way VLIW DSP with a traditional cache hierarchy. Texas Instrument DSPs can currently be found in Motorola Droid smartphones, Amazon Kindle E-book readers, and tablets like the Samsung Galaxy Tab 2. The Texas Instrument C67x DSP is described in further detail in Chapter 4.

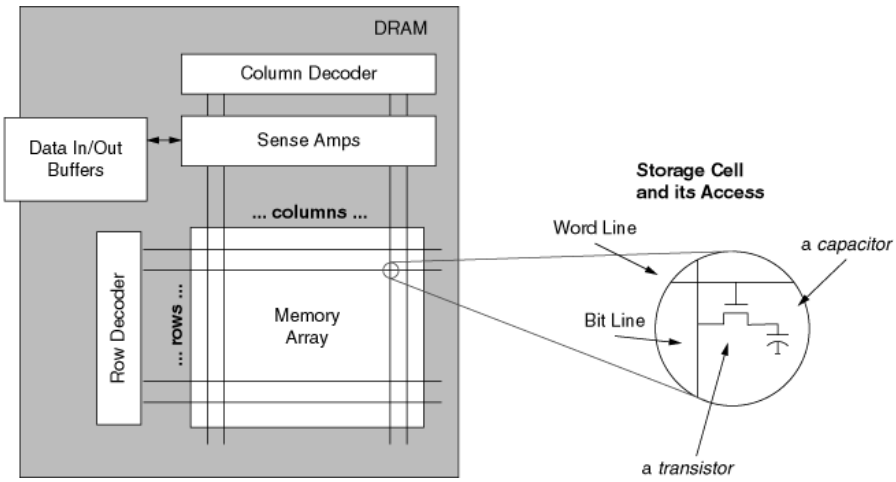

Figure 2.2: Organization of a DRAM memory array (from [22]).

2.3 Main Memory (DRAM)

Main memory or dynamic random-access memory (DRAM) is a type of memory, generally found off-chip that uses a single transistor and capacitor pair for each bit. DRAM is referred to as "dynamic" because the capacitors storing the bits of data leak their charge and so, to retain the stored information, each capacitor in the DRAM must be periodically refreshed (i.e. read and rewritten).

Each DRAM die contains one or more memory arrays, which contain a grid of storage cells with each cell storing one bit of data. Each memory array is arranged in rows and columns. To read or write to a specific location, the memory controller uses a row and column address to identify the intersection of a row and column within a memory array [22]. Specifically once a row is identified, an entire row of bit cells are brought into a row buffer. From the row buffer, the memory controller uses the column address to strobe the appropriate column and access the correct memory location. Figure 2.2 (taken from [22] with permission from Bruce Jacob) shows the breakdown of one DRAM memory array.

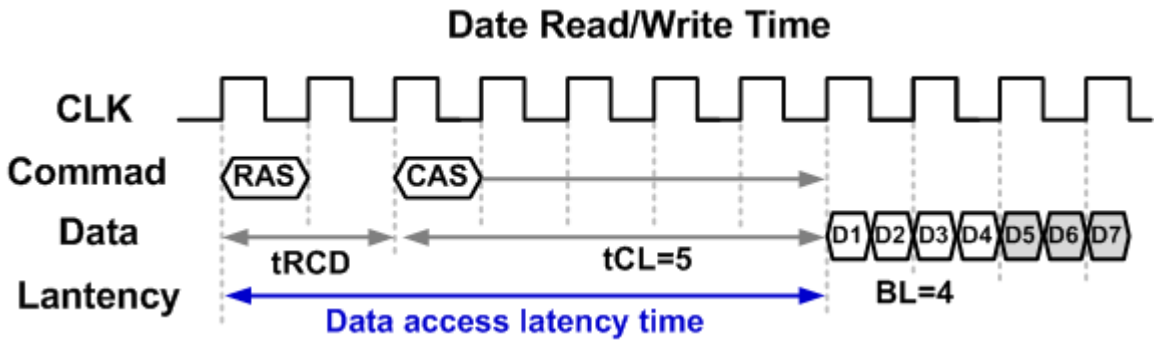

Figure 2.3: Timing diagram for a DRAM access.

The performance of main memory can be specified by a combination of several main memory parameters such as tRP, tRAS, tRCD, and tCL (or tCAS). The tRP is the row precharge time and represents the number of cycles needed to terminate access to a row in memory and open access to a different row. The tRAS is the number of cycles it takes to access a certain row of data after sending the row address. The tCL (or tCAS) is the number of cycles it takes to access a column of data after sending the column address. The tRCD (RAS to CAS delay) represents the number of cycles between opening a row of memory (tRAS) and accessing the columns (tCL or tCAS). Many times the tRCD delay can overlap with the tRAS delay. DRAM access latencies can vary widely based on if the data needed is in an already open row (tCL) or if the data resides in a different row and therefore must be opened and then read (tRCD + tCL,).

Figure 2.3 shows the read/write timing diagram for a DRAM access. To access memory, a row must first be selected and loaded into the row buffer. Each bit line is connected to a sense amplifier, which detects and amplifies the small voltage change of each bit line. The amplified signal is then output through a DRAM bus into the row buffer. The row is then active and columns are accessed for reading or writing. The dominant portion of the access time is determined by the row and column activate times plus CAS latency (tRCD + tCL) to allow

enough time for data to be streamed out [23] as shown in Figure 2.3. The term labeled “BL” in Figure 2.3 is the burst length, which represents how many “bursts” of data are read or written. For example, a BL value of 4 means that after the first column activation, an additional three memory accesses are performed from adjacent locations in the DRAM memory array, which are already in the row buffer. This memory access behavior improves the performance of applications that exhibit a degree of spatial locality because subsequent memory references will already be brought into the cache, which reduces the number of long latency main memory accesses.

3 Related Work

This chapter provides a survey of previous work. It includes a discussion of previous research in the areas of 3D memory design, 3D integration technology in high performance processors, 3D integration technology in domain-specific processors, and memory scheduling.

3.1 3D Memory Design

A number of studies have investigated 3D memory design and implementation without analyzing its impact on the full system [24] [25] [26] [27] [28] [29] [30].

Anigundi *et al.* [24] investigated different 3D DRAM design configurations that used wafer-level BEOL-compatible 3D integration techniques and TSVs. To accommodate the potentially significant pitch mismatch between DRAM word-lines/bit-lines and TSVs, they proposed two coarse-grained partitioning strategies for 3D DRAM designs that reduced the number of TSVs and relaxed fabrication constraints. Their proposed 3D partitioning design strategies reduced the silicon area, access latency and energy consumption when compared with 3D packaging with wire bonding and conventional 2D designs. To mitigate the yield loss of 3D integration, Anigundi *et al.* also proposed an inter-die, inter-sub-array redundancy repair approach to improve the success rate of memory repairs.

Tsai *et al.* [25] presented and validated a 3D cache delay and energy model called 3DCacti, which allows designers to partition the cache across different 3D device layers at various levels of granularity and explore different 3D options. Using their tool, the authors also explored a variety of 3D cache partitioning options and compared the impact of the delay and energy savings of different 3D cache partitioning strategies. They found that the savings in delay and

energy through 3D partitioning depends on the cache size, system requirements, the number of 3D device layers, and the technology node.

Wu *et al.* [26] proposed hybrid cache architectures composed of different memory technologies. They proposed both inter- and intra-cache level hybrid cache designs that combined SRAM, Embedded DRAM (EDRAM), Magnetic RAM (MRAM), and Phase-change RAM (PRAM) technologies. The motivation behind this cache design came from observing that caches made with different technologies offered dramatically different power and performance characteristics compared to the traditional SRAM-based cache. For example, although SRAM provided the shortest access latency, it consumed the most area and power while MRAM and PRAM technologies provide longer access latencies, but consumed less area and power. To take advantage of the best characteristics of each technology, they present a Hybrid Cache Architecture (HCA) that combines the technologies. They found an inter-cache-layer HCA design where individual levels in a cache hierarchy can be made of different memory technologies provided a 7% IPC improvement when compared to a traditional three-level SRAM cache design under the same area constraints. They also found that a more aggressive intra-cache-layer HCA design where a single cache level could be partitioned into multiple regions, each with a different memory technology, yielded a 12% IPC improvement over the baseline while still consuming the same area. Lastly, the authors note that since 3D integration technology enables the addition of more cache layers without increasing area, they present a two-layer 3D hybrid cache design using 3D technology that achieves an 18% IPC improvement over the baseline while reducing power by up to 70%.

Loh and Hill [27] [28] studied using 3D integration technology to make a multiple-layer, die-stacked DRAM cache. Their motivation for this 3D cache is that for many systems, the capacity

of stacked DRAM would not be large enough for all of main memory so they instead investigated a last-level DRAM cache. However, a 1 GB DRAM cache with 64-byte blocks would require 96 MB of tag storage; placing these tags on-chip would be impractical and putting them in the DRAM would be too slow. The authors reviewed multiple solutions including large cache line sizes (larger than the traditional 64 byte block) and sub-blocking. However, the authors noted that with large cache line sizes, there can be fragmentation problems where a small subset of data is used in a large cache line, which leads to wasted transfers; sub-blocking, on the other hand, increases the overhead of the tag array. To address these problems, the authors proposed a 3D DRAM cache design where the cache tags are stored along with data in the cache. They showed that scheduling the cache tag and data lookups as a combined operation makes this design practical. They also proposed a MissMap structure that allowed them to compactly track a very large amount of memory (i.e. 1 GB) using 2 MB of storage from the L3 cache reducing it from 8 MB to 6 MB. After combining the two techniques, they achieved 92.9% of the performance of an ideal 1 GB DRAM cache with an impractical 96 MB on-chip SRAM tag array.

Kim *et al.* [29] fabricated a 1 GB 3D-stacked Wide I/O SDRAM with four channels and 512 data pins using TSVs and a 50 nm technology node. The authors validated the connectivity of the TSVs through boundary scan tests and confirmed stable operation at 200 MHz using built-in self test (BIST). They also found the TSVs in Wide I/O 3D-stacked DRAM consumed 11.2x less I/O energy than off-chip metal wires.

Takaya *et al.* [30] presented a 3D test vehicle for wide I/O communication using TSVs. Their three-layer 3D stack was composed of a logic chip, active silicon interposer, and SRAM memory with a 4,096-bit data bus operating at 200 MHz. This test vehicle was fabricated with a

waveform capturer integrated on the silicon interposer to evaluate the signal integrity. The authors found their 4,096 wide I/O data bus could achieve 100 GB/s of memory bandwidth using a standard voltage supply of 1.2V.

Although my work also looks at 3D memory integration technology, I focus on analyzing the benefits of the technology on both a DSP processor and the entire memory hierarchy, rather than primarily studying main memory or the last level cache. The above works primarily focus on memory organizations and designs that are enabled with 3D integration technology with some works going so far as to fabricate their designs. Although one of my contributions (Chapter 6) investigates latency improvements via memory organization, it is not the primary focus of my dissertation. The primary contributions of my dissertation focus on the benefits of 3D technology on digital signal processors and proposing novel memory bandwidth scaling algorithms that take advantage of the increased main memory bandwidth that comes from using 3D DRAM with Wide I/O. Philosophically, I found [26] to be the most similar to my dynamic bandwidth scaling algorithms. Although they examined the memory hierarchy, their design approach of combining both fast access latency memories with low power consuming memories is similar to my own philosophy in proposing a dynamic bandwidth scaling algorithm that increases the main memory bus width during memory-intense phases and lowers the bus width during compute-intense phases to lower energy consumption.

3.2 3D Integration Technology in High Performance Processors

Prior studies have explored 3D integration technology for high-performance processors [31] [32] [33] [35].

Black *et al.* [31] investigated the benefits of dividing an Intel® Core™ 2 Duo microprocessor into two die and vertically stacking them atop each other to reduce the length and latency of the

wires between critical paths. They also studied increasing the size of the L2 through 3D integration and converting the L2 cache to DRAM technology for increased storage. Their vertical floor plan studies found that for a high-performance microprocessor, a 3D floor plan could reduce power by 15% while increasing performance by 15% for a 14°C increase in peak temperature. After voltage and frequency scaling, they found that the increase in temperature could be neutralized while still giving an 8% improvement in performance and a 34% reduction in power. Converting the cache to 3D stacked DRAM could also reduce the cycles per memory access by an average of 13%.

Liu *et al.* [32] explored the performance benefits of moving main memory DRAM on-chip and stacking it below a processor that is similar to an Alpha 21264. They reported that stacking DRAM below the processor and using TSVs reduced the main memory access latency by 60% and led to an average speedup of 13% for integer applications and 25% for floating-point applications. The 60% reduction in main memory access latency was cited as the primary reason for this performance improvement. They also explored the benefits of expanding the L2 cache with additional 3D layers and found that the peak performance for integer programs was achieved with an 8 MB L2 while floating-point program performance peaked with a 16 MB L2. This was due to the trade-off between fitting larger working sets and the increased access latency that comes with larger caches.

Similarly, Loh [33] investigated stacking 3D main memory DRAM below a high-performance Intel® quad-core Penryn microprocessor and also analyzed the performance impact of increasing main memory bandwidth by widening the data bus between main memory and the L2. He also extended previous work by considering more aggressive memory options and evaluated a “true 3D main memory” announced by the Tezzaron Corporation [34]. Loh argued

that previous studies did not fully exploit 3D stacking technology because the individual structures were still inherently two-dimensional. He evaluated this "true 3D main memory" where the individual bitcell arrays are stacked in 3D. This reduced the length of the internal buses, wordlines, and bitlines, which in turn reduced main memory access latency. This combination of optimizations provided an additional 32% improvement in main memory access time that was additive with any previous latency benefits, such as using TSVs instead of off-chip metal wires. His simulations showed that simply placing the DRAM on the same stack as the processor increased performance by an average of 34.7% on memory-intensive workloads, while expanding the bus of the 3D main memory architecture to 64 bytes further increased the average performance benefit to 71.8% over the 2D baseline. Loh also found that the "true 3D main memory" described above improved average performance by 116.8% over the 2D baseline. With these significant performance improvements in mind, he found the L2 miss handling architecture (MHA) was the new bottleneck and to address that, he proposed a new structure called a Vector Bloom Filter. This structure enabled a scalable MHA that complemented the increased capacity of the 3D-stacked memory system and provided an additional 17.8% performance improvement over his proposed 3D-stacked memory architecture.

Lastly, Kgil *et al.* [35] looked at Tier 1 server architectures and investigated using 3D integration technology to stack main memory DRAM below an eight-core system and found multiple stacked DRAM dies would be sufficient for a primary memory. This would allow them to remove the L2 cache and replace its area with an additional four cores (12 cores total). They found that a voltage and frequency scaled 12-core CPU utilizing 3D DRAM, but containing no L2 cache outperformed an eight-core CPU system with a large on-chip L2 and 3D DRAM by about 14% while consuming 55% less power. They also found that their architecture performed

comparably to an Intel® Pentium 4-like machine while only consuming about 10% of the power when running at a much lower voltage and frequency.

These studies demonstrated the potential performance and power benefits of 3D integration in the field of high performance microprocessors. Loh [33] in particular made significant contributions to the area. Although my dissertation also looks at the impact of 3D main memory on microprocessors, those processors are embedded DSPS, which have significantly different design constraints compared to the high-performance microprocessors described above. Although energy consumption is important in high performance processors, it is much more important on embedded DSPs that are usually found in mobile devices such as cell phones, which have limited battery life. These unique design constraints required my dissertation to focus on solutions that not only increased performance, but also targeted low energy consumption. This led me to propose my dynamic bandwidth scaling algorithms, which are described in Chapter 7. In addition to the underlying processor architecture being different, my work also differs due to the applications I evaluated, which were primarily multimedia-based applications commonly run on mobile computing devices. These applications are described in Chapter 4.

Lastly, some of the performance claims in the above works are contingent upon 3D stacked memories and TSV interconnects providing a significant reduction in main memory access latencies (as high as 60% as quoted in [32]). Using TSVs instead of off-chip metal wires is cited as the primary reason for this latency reduction. These latency reduction claims are used in many works and unfortunately, we disagree with this latency reduction claim. As described in Chapter 5, simply stacking 3D DRAM below the processor and using TSVs does not provide as large of a latency savings. This finding reduces some of the performance benefits in the above studies. However, my work does not refute the performance claims of the “true 3D main memory”

described in [33]. It instead looks at stacking 3D DRAM below the processor and using TSV interconnects since that is the commonly studied case in 3D memory research.

3.3 3D Integration Technology in Domain-Specific Processors

There have also been a number of studies that investigated 3D integration technology in domain-specific systems [36] [37] [38] [39] [40].

Al Masshri *et al.* [36] explored the benefits of using 3D stacked caches on GPUs, and compared 3D stacked caches using MRAM (Magnetic Random Access Memory) and SRAM technologies. Their results showed that by using 3D stacked caches, GPUs achieved a 53% geometric mean speedup on 3D graphics and gaming benchmarks. They also found that replacing the SRAM caches with MRAM technology yielded performance degradation, but provided power benefits, making it more appealing for power-conscious applications.

Pan *et al.* [37] investigated 3D memory integration on VLIW DSPs. Their work is the most similar to my dissertation work. They explored the benefits of stacking main memory DRAM below a DSP and then converted the L2 from traditional SRAM technology to their own multi-Vth DRAM design, which was much faster than traditional DRAM. They found that using 3D integration technology to migrate main memory into the package yielded a 10% to 80% IPC improvement over the baseline. They also found that by using their own multi-Vth DRAM design for the L2 (instead of traditional SRAM technology), they could double the size of the L2 (without impacting area) and see an additional 10% improvement in IPC.

Sun *et al.* [38] proposed a 3D memory hierarchy where three SRAM layers could be stacked below a multi-core DSP and configured as instruction or data caches for each DSP core. They found that this memory architecture could improve the system performance of a single-channel H.264 decoder by 30% and a multi-channel H.264 decoder by 54% compared to the 2D memory

baseline. They also found this memory architecture could increase the performance of a JPEG decoder by six times by placing the data section (heap and stack structures) in the 3D SRAM instead of the off-chip DDR2 memory.

Kim *et al.* [39] presented a reconfigurable heterogeneous multimedia processor for various embedded applications on a handheld device. Their implemented processor included a custom-designed I/O interface circuit with reconfigurable output drivers on a silicon-interposer channel to achieve 8x higher memory bandwidth and found it achieved a 1.7x speedup in full augmented reality processing.

Sampson *et al.* [40] implemented a system architecture and specialized accelerator unit for low-power 3D ultrasound beam formation, which they call Sonic Milip3De. Since 3D ultrasound offers unique challenges for computing due to its computational complexity and low power requirements due to its close contact with human skin, the authors leveraged 3D die stacking technology to split the analog components, analog-to-digital converters, SRAM storage, and a 1,024 unit beamsum accelerator array across three silicon layers for shorter wires and a compact design. The authors also developed a new iterative delay calculation algorithm to exploit data locality and lower power. With these optimizations, their 3D design was able to image a fully sampled array within a 16W full-system power budget, which was over 400x less power than a DSP-based solution. The authors estimate that with current scaling trends, their system will meet the 5W target power budget for safe use on humans by the 11 nm technology node.

Although [37] and [38] are the most similar to my dissertation research, I conducted a more thorough design space exploration by considering a wider range of 3D memory configurations. I specifically targeted memory configurations that increased the main memory bus width, which were not considered in [37] and [38]. Furthermore, some of the performance results in [37] and

[38] rely on the original latency claims of 3D main memory (45% to 60% latency savings), and as will be explained in Chapter 5, I show this claim to be inaccurate. Therefore, I believe my dissertation research to have more accurate performance results. Lastly, my dissertation research also looks into dynamic bandwidth scaling algorithms, which none of the above works consider. To the best of my knowledge, these topics have yet to be addressed or thoroughly studied for DSP systems.

3.4 Memory Scheduling

There have been a number of studies on main memory scheduling algorithms, memory controllers, and memory bandwidth [41] [42] [43] [44] [45] [46] [47].

Rixner *et al.* [41] analyzed the benefits of various algorithms for reordering memory accesses for streaming applications. They describe a number of memory scheduling policies including in-order, priority, open, closed, most pending, and fewest pending. The in-order scheduler takes the oldest pending DRAM reference. A priority scheduler performs the pending memory reference with highest priority. An open scheduler precharges a bank only if there is no pending references to other rows in the bank and there are no pending references to the active row. This policy is employed when there is significant row locality, making sure future references that target the same row are serviced quickly. A closed scheduler precharges a bank as soon as there are no more pending references to the active row. It should be employed if it is unlikely that future references will target the same row as previous references. A most pending scheduler selects the row or column access to the row with the most pending references; it can be used to prevent starvation. Lastly, the fewest pending policy selects the column access to a row targeted by the fewest pending references, which minimizes the time a row with little demand remains active. Based on these observations, they proposed an in-order, first-ready access scheduler that

considers all pending references and schedules a memory operation for the oldest pending reference. They found that the in-order, first ready access scheduler improved performance by an average of 17% and improved bandwidth utilization by 40%. They also proposed an aggressive reordering access scheduler increased locality and concurrency to improve performance by 30% and bandwidth utilization by 93%. This work was one of the first to propose aggressive main memory scheduling algorithms, which are now commonly used in modern memory systems.

Hur *et al.* [42] presented an adaptive history-based (AHB) scheduler that used the history of recently scheduled memory operations to allow the scheduler to 1) better utilize the delays associated with its scheduling decisions and 2) select operations that match the program's mixture of Reads and Writes to help alleviate bottlenecks within the memory controller. The authors found their memory scheduler improved performance on an IBM Power5 by 15.6%, 9.9%, and 7.6% for the Stream, NAS, and commercial benchmarks, respectively and argue that as memory traffic increases, the benefits of their AHB scheduler would also increase.

Ipek *et al.* [43] proposed a self-optimizing memory controller design that uses the principles of reinforcement learning (RL) to make memory scheduling decisions. Their RL-based memory controller's goal was to learn the optimal memory scheduling policy and associate the system state with long-term performance impact so as to schedule the memory operation that it estimated will produce the highest long-term reward (performance). By continuously updating the long-term reward values, the RL-based memory controller could adjust to changes in workloads in real-time. The authors found their RL-based memory controller improved the average performance of a four-core CMP by 19% and the DRAM bandwidth utilization by 22% compared to a modern memory controller.

Leibowitz *et al.* [44] presented a mobile memory interface that utilized a globally synchronous clock pause, which allowed the memory controller to synchronously pause the entire memory interface to gate dynamic power consumption. This approach allowed the DRAM to operate in a burst mode that could match the nature of the memory traffic while reducing power consumption during idle periods. A test-chip was fabricated and achieved a peak memory bandwidth of 4.3 GB/s at 3.3 mW/GB/s power efficiency, which could be scaled to 12.8 GB/s for more aggressive memory and power demands.

David *et al.* [45] evaluated dynamic voltage and frequency scaling for memory bandwidth scaling. They observed that in a typical server platform, memory consumed on average 19% of the system power and that although increasing compute cores increases the demand on memory, many workloads do not require maximum memory bandwidth. They found that reducing the memory frequency incurred minimal performance degradation while reducing memory frequency also allowed them to reduce the memory voltage. Based on these observations, the authors proposed a dynamic voltage and frequency scaling algorithm that is based on memory bandwidth utilization and found their algorithm reduced average memory power by 10.4% with a performance degradation of only 0.17%.

Yoon *et al.* [46] proposed and evaluated an adaptive granularity memory system that combined fine-grained and coarse-grained memory accesses. Modern memory systems use coarse-grained memory accesses to bring an entire L2 cache line with multiple main memory bursts. However, the authors noted that during code segments where spatial locality is low, this coarse-grained approach was inefficient. The authors implemented an adaptive granularity memory system with sector caches and sub-ranked main memory that could go between coarse-grained and fine-grained memory accesses and found it could improve the performance of

memory-intensive applications by 61% without ECC (error correcting codes) and 44% with ECC. They also found their adaptive granularity memory system could lower memory power consumption by 29% without ECC and 14% with ECC.

Deng *et al.* [47] presented a main memory power management scheme called MemScale, which applies dynamic voltage and frequency scaling (DVFS) to the memory controller and dynamic frequency scaling (DFS) to the memory channels and DRAM devices to improve memory energy consumption. MemScale was controlled by an operating system policy that determined the DVFS/DFS settings for the memory system based on the memory bandwidth needs, potential energy savings, and the performance degradation the applications could withstand. Unlike traditional low-power memory states that require entire DRAM ranks to be idle, MemScale's modes are active and do not rely on memory idleness. Through simulation the authors found MemScale could reduce memory energy consumption between 17% and 71% with a maximum performance degradation of 10%.

These works demonstrated the performance and energy benefits of advanced memory scheduling algorithms, memory controller optimizations, and dynamic voltage and frequency scaling on main memory. Although one of the contributions of my dissertation is proposing dynamic bandwidth scaling algorithms to lower DSP energy consumption, which was the goal of [44], [45] and [47], my proposed algorithms do so through a different mechanism. My dynamic bandwidth scaling algorithms vary the main memory bus width to lower main memory energy consumption while the authors in [44], [45], and [47] use a global pause or dynamic voltage and frequency scaling. Similarly, adaptive granularity memory accesses proposed in [46] is also similar to my dynamic bandwidth scaling algorithms. The authors of this study also dynamically change main memory accesses, but do so with changes to the main memory and cache

organizations. Furthermore, their coarse-grained granularity is no wider than typical main memory accesses (one L2 cache line size) while my dynamic bandwidth scaling algorithms can fetch up to three additional L2 cache lines through the use of TSVs, which is only possible when utilizing 3D integration technology.

Additionally, while [42] and [43] use previous memory requests to make current memory decisions and are similar to how I designed my algorithms (described in Chapter 7), one of my algorithms considers DSP performance (IPC) to make its bandwidth scaling decisions. Both studies also focused on improving performance and bandwidth utilization while my algorithms focus on efficiently using memory bandwidth so as to reduce DSP energy consumption while maintaining performance. Lastly, my dissertation research focuses on 3D memory technology and specifically on a key advantage of 3D memory technology (increased main memory bandwidth) as its motivation and none of the above studies considered 3D memory technology and instead considered more traditional memory architectures. To the best of my knowledge, my dissertation research is the first to propose dynamic bandwidth scaling algorithms for 3D memory systems utilizing significantly increased main memory bus widths.

4 Experimental Framework

This chapter provides a detailed description of the simulation environment I used for DSP performance and energy evaluation. It also includes a more detailed description of the baseline DSP I used for comparison, which is similar to a Texas Instrument C67x DSP. Lastly, this chapter concludes with a description of the six benchmarks I used for evaluation.

4.1 Simulation Environment

4.1.1 Trimaran

For performance evaluation, I used the Trimaran simulator [48] running within the EPIC-explorer framework [49]. The Trimaran simulator is a cycle-accurate, highly parameterizable integrated compilation and performance evaluation tool used to evaluate embedded and high-performance VLIW architectures. Trimaran is comprised of the following:

1. A parameterized instruction-level parallelism (ILP) architecture called HPL-PD [50].
2. A machine description language for conveying HPL-PD architectures.
3. An easily modifiable optimizing compiler that employs an extensible IR (intermediate program representation), which has both an internal and textual representation. The IR supports modern compiler techniques such as representing data and control dependencies and control flow.
4. A HPL-PD architecture simulator, which can be parameterized via a machine description. This simulator provides run-time information on execution time that can be used for profile-driven optimizations.

HPL-PD adopts a philosophy where the compiler is responsible for statically scheduling the execution of a program and therefore the compiler needs exact information about the architecture

including the register file structure, included operations, resources in the architecture, resource utilization patterns, and operation latencies. A machine-description (MDES) database specifies these parameters, and the architectures are defined using a human-readable machine description language, called HMDES [51].

The Trimaran simulator is comprised of three components: the OpenIMPACT compiler, the Elcor compiler, and the Simu simulator. OpenIMPACT takes the application's source code and the description of the VLIW processor as inputs and performs compilation and static scheduling of the operations. The Trimaran simulator uses the OpenIMPACT compiler to compile the original source code into an assembly intermediate representation (IR) called Lcode that is optimized for Instruction-Level Parallelism (ILP), but not targeted towards any specific architecture. The Elcor compiler takes the Lcode and a machine-description that specifies the machine and compiles the code for the target machine. During the Elcor step, the benchmark is profiled and a binary is generated by linking with the emulation library (Emulib), which provides the code needed to simulate each of the instructions scheduled by Elcor. The generated binary is then simulated on the Simu simulator and execution statistics are captured. These statistics portray the performance behavior of the benchmark on the target VLIW architecture.

Additionally, as noted earlier, the Trimaran simulator was run within the EPIC-explorer framework. EPIC-explorer is a parameterized VLIW-based platform containing area and energy estimation models and is primarily used for design exploration. For the processor, EPIC-explorer's energy estimation models are based on the Cai-Lim model [52] and adapted to VLIW processors. This energy estimation model subdivides the processor into a set of functional blocks called FBUs (Functional Block Units) and gives two measurements for each type of circuit: active energy density and inactive energy density where the inactive energy density is set to 10%

of the active energy density. For the memory hierarchy, EPIC-explorer uses the analytical model in CACTI [53], which includes energy components for the inputs, outputs, bitlines, and wordlines. For the wider bus widths simulated in Chapter 6, this model does not take into account the extra TSVs and their increase in leakage power. EPIC-explorer has been integrated with Trimaran and takes the execution statistics and a configuration file obtained from the Trimaran simulator and uses them to make energy estimations for each functional block.

Lastly, the version of the Trimaran simulator I used (version 4.0) provides an interface for simulating the memory hierarchy through an optional simulator called M5elements. M5elements is a cache simulator that allows the Simu simulator to use the memory subsystem of the M5 simulator [54] to gather detailed memory statistics.

For this dissertation, modifications were made to both Trimaran and EPIC-explorer including the addition of an energy estimation model for the main memory bus. This energy estimation model takes the energy per I/O transfer derived in [29] for both 2D metal wires and 3D TSVs, the number of DRAM accesses, and bus width to calculate the energy consumed by the main memory bus. Modifications were also made to Trimaran to support implementation of my proposed bandwidth scaling algorithms and adjacent cache prefetching including the addition of more main memory and cache statistics. Trimaran was chosen for my dissertation research because, at the time, it was one of the only simulators for VLIW processors. Additional features such as an integrated energy and area model from EPIC-explorer also made it an attractive simulation platform. All of the processor and memory configurations as well as my bandwidth scaling algorithms were simulated on the Trimaran simulator and all performance and energy results reported in Chapters 5, 6, and 7 were generated using Trimaran, M5elements, and EPIC-explorer.

4.1.2 CACTI

The CACTI memory simulator [53] was used to aid in the modeling of the timing and energy consumption of traditional DRAM. CACTI provides an analytical model for cache and memory access times, cycle times, area, leakage power, and dynamic power. By having area, timing, and power models integrated together, CACTI allows users to weigh the benefits and tradeoffs of their memory designs, which allows designers to have a better understanding of the impact of different design choices. CACTI accepts a very detailed set of memory parameters, which allows the user to explore a wide variety of memory configurations. CACTI has been rigorously updated throughout the years including improvements in accuracy and support for smaller technology nodes. The fifth major update to CACTI added the ability to model DRAM.

For my dissertation research, CACTI was used to model a 512-MB 2D DRAM with a page size of 8,192 bytes and one read-write port in 65-nm technology, which is a common main memory configuration for mobile phones. This configuration had a random access latency of 56 ns, which is very close to the latency found in Micron's DDR2 specification sheet [23] for a memory with the same configuration. However as noted in Chapter 2, not all DRAM accesses are random and many of them do not require a row activation time (tRAS) and simply need a column strobe time (tCAS) since the memory request is already in the row buffer. This leads to faster access latencies for many DRAM requests. For that reason, CACTI was primarily used to generate DRAM timing parameters such as row activation time (tRAS), column strobe (tCAS or tCL), precharge time (tRP), etc., which was then used in the trace-based memory simulator to generate a more accurate DRAM access latency. This trace-based memory simulator is described in the next section.

4.1.3 DRAMSim2

To obtain more accurate main memory access latencies, DRAMSim2 [55] was used. DRAMSim2 is a cycle-accurate DDR2/3 DRAM simulator that models a detailed and accurate memory controller that issues commands to a set of DRAM devices attached via a traditional memory bus. DRAMSim2 fills the void in cycle-accurate memory simulators. In particular, many CPU simulators significantly underestimate the effects of the memory system by using overly simplistic models of the memory system. These simple memory models fail to take into account highly complex behaviors of modern memory systems such as request reordering and open and closed page policies, which DRAMSim2 models. To validate its accuracy, DRAMSim2 includes Verilog timing models for Micron DDR2/3 DRAM parts. Whenever DRAMSim2's memory controller issues a command, this command is first executed on the Micron Verilog model with the same timing parameters. This is done to determine if the DRAMSim2 memory controller has violated any timing requirements.

DRAMSim2 includes an accurate memory controller model and issues individual memory requests, unlike CACTI's analytical model. It is a trace-based memory simulator that takes a trace of memory requests from an application and accurately models each individual memory request and its interaction with other memory requests in the memory transaction queue (i.e. memory request reordering). Once the trace has been analyzed, results are reported, which include the average latency of requests within a timing window, how many requests went to each DRAM bank, etc.

For my dissertation research, I modified DRAMSim2 to output more fine-grain timing information for each memory request. I then modified the system.ini file with the timing parameters obtained in CACTI and discussed in Section 4.1.2. These parameter modifications to

the system.ini file ensured that DRAMSim2 would simulate the same DRAM as the one modeled in CACTI. I then modified the Trimaran simulator to gather and output each individual main memory request, which I then parsed into the format required for a DRAMSim2 memory trace. Once the memory traces for each of my applications (described in Section 4.3) were generated, each memory trace was run on DRAMSim2 and its timing information collected.

As expected, each application had a faster average latency than the random access latency reported in CACTI in Section 4.1.2 (56 ns). As discussed earlier, this was because the memory behavior of these applications was not random and the applications exhibited some spatial locality. This meant in an open page main memory policy, some of the memory requests did not need a row activation (tRAS) and only needed a column strobe (tCAS or tCL), which would lower the memory request's access latency. For our six applications, we found the average main memory latencies to be between 38 and 40 ns. These were the main memory latencies used in my Trimaran simulations with a 2D DRAM memory system.

4.2 Baseline Processor

For my dissertation research, the baseline DSP architecture I used for evaluation is similar to a Texas Instrument C67x series floating-point processor [21]. The C67x is designed to be a low-power application DSP. It is a dual-core VLIW processor running at 1 GHz and has separate L1 caches for data and instruction and a shared L2 cache. The C67x DSP has two paths that can issue four instructions each for a total of eight instructions per cycle. All integer execution units are 32-bits wide, and operations are statically scheduled by the compiler and executed. Figure 4.1 shows a block diagram of the Texas Instrument C67x DSP. Table 4.1 summarizes the Texas Instrument C67x configuration.

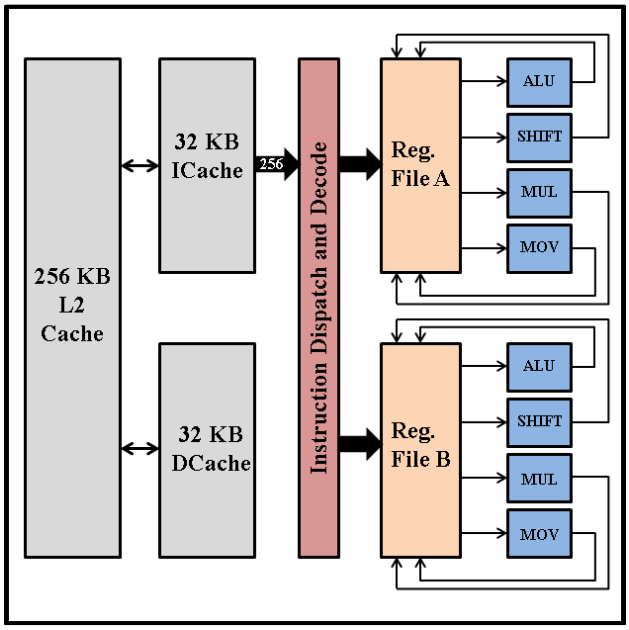

Figure 4.1: Block diagram of a Texas Instrument C67x DSP.

Table 4.1: Texas Instrument C67x Configuration.

Parameter	**Setting**
Frequency	1 GHz
L1 data cache	32-KB, 2-way, 512-bit lines, 1 cycle
L1 instruction cache	32-KB, 1-way, 256-bit lines, 1 cycle
Unified L2 cache	256-KB, 4-way, 1,024-bit lines, 7 cycles
# of ALUs	6
# of multiply units	2
# of memory units	1
Main memory size	512-MB
Main Memory bus width	64 bits
2D main memory latency	38 to 40 ns

4.3 Benchmarks

To evaluate memory configurations and bandwidth scaling algorithms, I used applications from the MediaBench benchmark suite [56], the SPEC CPU2006 benchmark suite [57], and the San Diego Vision Benchmark Suite [58]. The chosen applications are commonly run on handheld mobile devices and encompass a variety of signal processing application types including multimedia processing, 3D graphics, and speech recognition. The six benchmarks I used for my evaluations were:

1. **Unepic** is an experimental image decompression utility (EPIC) found in the MediaBench benchmark suite. It is a lossy image decompression algorithm, meaning that the output image is not exactly identical to the input image. Unepic has been designed with filters to allow for fast decoding without floating-point hardware. Unepic was specifically chosen for evaluation since image decompression is a common task done on mobile phones when rendering webpages.
2. **482.sphinx3** is a speech recognition system developed at Carnegie Mellon University [59] and is part of the floating-point portion of SPEC CPU2006. Speech recognition converts spoken words from audio to text by reading the input (audio) and then repeatedly processing it with different "beams," which are the probabilities used to prune the set of active hypotheses during each recognition step. Speech recognition was chosen for evaluation because it is a complex application that is becoming an important portion of popular consumer applications such as personal assistants like Siri found on Apple's iPhone [60].
3. **Disparity** is a 3D reconstruction application in the San Diego Vision Benchmark suite that is based on Stereopsis [61], also known as Depth Perception. Disparity takes a pair of

stereo images, taken at slightly different positions and computes the depth information of the objects represented in both pictures. The depth information of a scene gives the relative position of the objects, which vision systems can then use for cruise control, pedestrian tracking, and collision detection. Although originally intended for robot vision systems found in cars and unmanned aerial vehicles (UAVs), computer vision is important for smart phones as complex Augmented Reality (AR) applications become more popular among consumers. For these reasons, Disparity was chosen as one of benchmarks for evaluation.

4. **Mser** is an object detection application that takes a digital image and determines the location and size of human faces (i.e. face detection). The San Diego Vision implementation of this application is based on the Viola Jones Face Detection algorithm [62]. The granularity of the algorithm's operations occurs at the pixel-level and the computations performed are complex. Face detection is found in biometrics, video surveillance, human computer interface (HCI), and image data management. Like the other SD-VBS applications, it was chosen for evaluation because of its wide-spread usage in consumer mobile devices.
5. **Stitch** is an image or photo stitching application found in the San Diego Vision Benchmark suite. The implementation of the Stitch benchmark is based on [63] and uses the RANSAC algorithm [64] for image matching. Stitch combines multiple photographic images with overlapping fields of view into either a segmented panorama or high-resolution image. Image stitching is found in photography and movie making applications and was chosen for evaluation due to its use in camera applications found in mobile phones.

6. **Tracking** is a feature tracking application used to extract motion information from a sequence of images. This is done with feature extraction [65] and a linear solver that calculates the movement of the extracted features. This San Diego Vision benchmark implements the Kanada Lucas Tomasi (KLT) algorithm [66] for feature tracking, which is comprised of three major computation steps: image processing, feature extraction, and feature tracking. Image processing is done at a pixel level granularity while feature extraction and tracking, the core of the algorithm, operates at a coarse grained level based on what features are identified. This application has uses in the automotive industry, robotic vision, and video surveillance, but is also finding uses in consumer electronics such as Microsoft's gaming peripheral Kinect [67] and object detection in cameras.

As noted above, these applications were chosen since they were representative of the types of applications commonly run on modern handheld mobile devices such as smartphones. Some of the applications such as image compression are relatively simple and have been in use for years, but many of them such as face and speech recognition are becoming more essential towards the end-user experience on consumer mobile devices. These types of applications are not only compute intense, but also memory intense making them well-suited for the 3D memory research detailed in this dissertation. It is my belief that future mobile computing applications will continue this trend and become more complex and as such, the chosen benchmarks are representative of the types of applications that will be run on future DSP systems.

5 Reevaluating 3D Main Memory

In this chapter, I present a new model for 3D main memory latency and show through accurate circuit-level simulations that the original claim that 3D main memory reduces access latency 45% to 60% is inaccurate. With this model, the latency reduction is no more than 2.4 ns (or 4.1%)[1]. This is a significant finding since many of the performance improvements found in 3D memory studies were contingent upon this memory access latency reduction. In this chapter, I also re-evaluate the performance benefits of 3D main memory with this new model and find that the one of the advantages of 3D main memory needs to be re-examined.

5.1 A Model for 3D Main Memory

As described in Chapter 3, there have been a number of studies using the claim that going to 3D main memory and using TSVs instead of off-chip metal wires results in a 45% to 60% reduction in main memory access latency, which increases the performance of applications with high L2 miss rates that access main memory often. However, regardless of 2D or 3D technology, the DRAM bank organization remains the same as described in Section 2.3. Both 2D and 3D DRAM arrays are composed of rows and columns requiring the same timing parameters such as tRAS and tCL.

Therefore, if the DRAM bank organization and data output pipelining architectures remain the same, which is likely due to the strong desire for low cost per bit by the DRAM industry, the key difference in 2D and 3D DRAM memory access time is the difference in interconnect latency between the 2D and 3D buses for clocks, data and commands; specifically the latency difference between off-chip metal wires and TSVs. Figure 5.1 illustrates our 2D and 3D physical

[1] This new model for 3D main memory was developed by collaborating with Professor Gyung-Su Byun at West Virginia University.

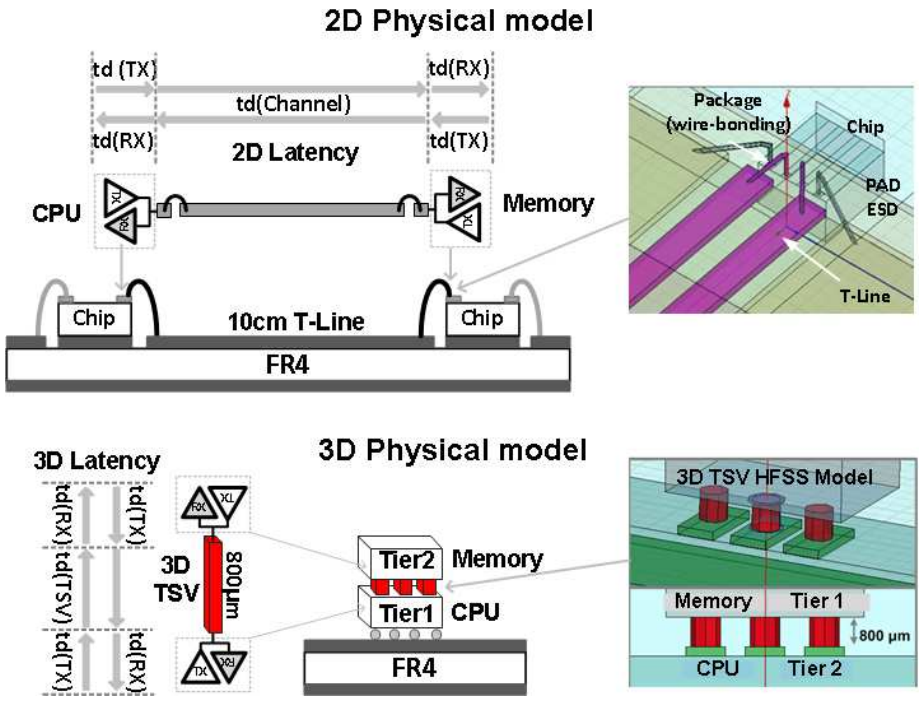

Figure 5.1: 2D and 3D main memory physical interconnect models.

interconnect models for the memory I/O bus with the 2D physical model on the top and the 3D physical model below it. As can be seen in the figure, both models are composed of three key components: 1) a transmitter (TX), 2) a receiver (RX), and 3) a transmission medium or channel (metal wires for the 2D case and TSVs for the 3D case). The transmitted signal is degraded by the frequency dependent attenuation of a band-limited channel. For our 65nm CMOS 2D and 3D physical bus models, a push-pull transmitter and single-ended differential amplifier receiver are used because they provide the most general transceiver architecture [68].

To obtain highly accurate latency results for 2D and 3D transmission medium, 2D FR-4 and 3D TSV channels [69] were modeled using the 3D full-wave electromagnetic field simulator, HFSS (High Frequency Structure Simulator) [70]. The parameters for the 2D and 3D channel models are listed in Table 5.1. The 2D memory channel model included a printed circuit board (PCB) channel of 10 or 20 cm long in FR-4 substrate, a standard FCBGA package for the

memory controller, a 4mm-long Chip Scale Package (CSP) wire bond package for the memory devices, solder balls, capacitive loadings for electrostatic discharge (ESD), as well as device capacitance and terminations from the transmitters and receivers. However, the 3D channel included only short, vertical TSV wires between tiers [71] that are 0.8mm in the worst case and sometimes can be reduced from 0.8 mm to 0.1 mm by thinning each die [69].

In general, DRAM access latency includes the time to request data from the DRAM core (memory controller latency), the time to retrieve the data from the DRAM core, and the interconnect delay through the bus interface unit. In Table 5.1 the 2D column represents traditional off-chip metal wires, the 3D column is the same as the 2D column except the interconnects are replaced with TSVs, and the 3D–Aggr column is the case where we aggressively removed transmitters from the memory controller and receivers from memory devices to further reduce the core access time for the 3D channel due to very short TSV interconnect (no delays for long 2D traces, packaging, ESD components and much less capacitive loading). Therefore, 3D-Aggr includes only address multiplexing due to on-chip TSV interconnects.

Table 5.1: 2D vs. 3D Channel Model Parameters.

Parameters	**2D [71]**	**3D [69]**	**3D-Aggr [69]**
Traces length	200 mm (Microstripe)	0.8 mm (short TSV)	0.8 mm (short TSV)
Capacitive Loading (/w ESD components)	1 pF	0.01 pF (TXRX Cgate)	0 pF (TXRX removed)
FC-BGA Solder balls	650 um	No solder balls	No solder balls
CSP Wirebonds	4 mm	No Wirebonds	No wirebonds
Transmitter and Receiver	304 ps	289 ps	0 ps
Impedance of channel	50 ohm	50 ohm	50 ohm
Terminations	50 ohm	50 ohm	50 ohm
Interconnect Latency	**1,522 ps**	**310 ps**	**21 ps**

In prior works, the off-chip channel (i.e. metal wires) latency was assumed to be 10 ns in 2D DDR SDRAM [72]. However, as can be seen in Table 5.1, if the architecture of the DRAM internal core, address multiplexing and data I/O transceiver does not change, the latency decrease between 2D and 3D-Aggr (transceivers of the MC and memory devices are removed) is only 1.5 ns. We validated these latency differences via simulation with the Spectre circuit simulator [73]. Table 5.2 shows the latency results from those simulations, which are based on accurate circuit and physical models including a 2D package (wire-bonding), ESD components, and standard I/O pad parasitic components.

Table 5.2: Transmitter, Receiver, and Channel Latencies.

Component	**2D T-line (10 cm)**	**2D T-line (20 cm)**	**3D TSV**
td (TX)	69 ps	70 ps	69 ps
td (channel)	620 ps	1,210 ps	21 ps
td (RX)	235 ps	242 ps	220 ps
Interconnect Latency	**924 ps**	**1,522 ps**	**310 ps**

When taking into account the DRAM access time, these models show the reduction in total access latency (DRAM access latency plus interconnect latency) going from 2D to 3D is as little as 0.6 ns (0.9 ns to 0.3 ns) and no more than 1.2 ns (1.5 ns to 0.3 ns). When this value is doubled to account for each access going to and back from the memory, the savings is as little as 1.2 ns and no more than 2.4 ns. For example, the total 2D access latency for a random access is 59.0 ns, where 56.0 ns comes from the random access latency for a DDR2 DRAM [23] and 3.0 ns (1.5 × 2) comes from the off-chip channel latency. Similarly, the total 3D access latency for a random 3D memory access is 56.6 ns (56.0 ns + 0.3 ns + 0.3 ns), which corresponds to a reduction of about 4.1% (2.4 ns / 59 ns). However, since our baseline processor does not implement critical word first, the processor must wait for the entire L2 cache line to be filled. Since the L2 cache has 1,024-bit lines this means the first 64 bits take 56 ns and the 15 additional bursts (of 64 bits)

are used to fill the rest of the cache line. Since each additional burst requires 1 cycle (or 1 ns) [23], this makes the total latency 71 ns (56 ns + 15 ns). For this scenario, TSVs reduce the access latency by 3.4% (2.4 ns / 71 ns). As indicated earlier, if the DRAM bank organization is reorganized, as shown in [33], the reductions would be larger. However, it is widely cited that simply going to 3D and using TSVs instead of off-chip metal wires causes a 45% to 60% reduction in overall latency. Yet, through simulations it was found that the latency improvement from using TSVs is incorrect, which justifies a reexamination of some of the results that have been published in the area of 3D memory integration.

5.2 Reevaluating 3D Main Memory on DSPs

In this section, the details of my experimental methodology are explained and then a performance analysis of our proposed 3D main memory model is presented. This performance analysis re-examines the potential performance benefits of 3D memory integration on embedded DSPs.

5.2.1 Methodology

I began by simulating the baseline C67x-like DSP detailed in Table 4.1 to establish the performance of the baseline DSP using traditional 2D main memory. I then simulated the baseline DSP and reduced the main memory latencies to 21 and 22 ns (depending on the application) using the original, more aggressive 3D main memory latency reduction claim (45%). These two simulations allowed me to quantify the potential performance benefits for the original 3D main memory latency reduction claims in isolation. I then simulated the same baseline architecture with the more accurate 3D latency savings calculated in Section 5.1 and instead reduce the main

memory latencies from 38 and 40 ns to 36 and 38 ns, respectively, to show the actual performance benefits. Table 5.3 summarizes our simulation parameters.

Table 5.3: Texas Instrument C67x Simulation Parameters.

Parameter	**Setting**
Frequency	1 GHz
L1 data cache	32-KB, 2-way, 512-bit lines, 1 cycle
L1 instruction cache	32-KB, 1-way, 256-bit lines, 1 cycle
Unified L2 cache	256-KB, 4-way, 1,024-bit lines, 7 cycles
# of ALUs	6
# of multiply units	2
# of memory units	1
Main memory size	512-MB
Main Memory bus width	64 bits
2D main memory latency	38 to 40 ns
Original 3D DRAM latency	21 and 22 ns
Accurate 3D DRAM latency	36 to 38 ns

5.2.2 Results

As described above, we begin by comparing the baseline configuration with 2D and 3D main memory. Figure 5.2 shows the performance of each benchmark, with all results normalized to the baseline 2D configuration. With 3D main memory having latencies of 21 and 22 ns (Baseline (3D)), we observe a performance improvement of 6.5% (using the geometric mean) over the baseline. The unepic, sphinx, disparity, mser, and tracking benchmarks exhibit the largest improvements (3.3% to 15.1%). These performance improvements can be attributed to these benchmark's higher L2 miss rates (10.0% to 52.2%). Since these benchmarks access the main memory more frequently, they benefit more from the lower main memory access latency. However, as demonstrated in Section 5.1, this lower main memory access latency claim is

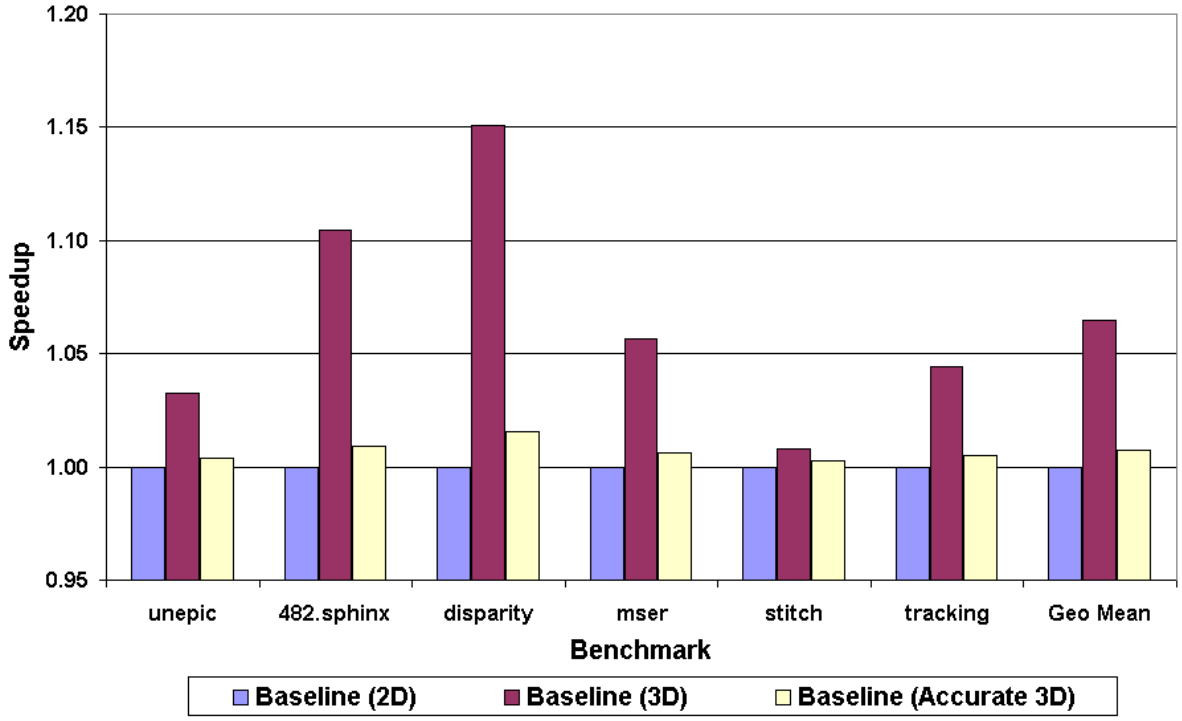

Figure 5.2: Performance results of the six benchmarks on the baseline processor with 2D and 3D main memory.

inaccurate. Since this claim is the basis for many 3D memory-based studies, however, this set of experiments was designed to quantify the performance benefit that could be realized if this claim were accurate.

Figure 5.2 also shows the performance results for our baseline system with a 3D memory system using the latency reduction of 2.4 ns calculated in Section 5.1 (Baseline (Accurate 3D)). As expected, such a small reduction in main memory latency results in the benchmarks performance being nearly the same as the baseline 2D configuration. With the accurate 3D main memory latencies, we observe a performance improvement of only 0.7% (using the geometric mean) over the baseline. Three benchmarks (sphinx, disparity and mser) exhibit a small increase in performance, but this improvement is less than 2%.

These results show that the small latency reductions of 3D main memory through the use of TSVs leads to almost no performance benefit for our C67x-like DSP and that one of the key benefits of 3D main memory needs to be re-evaluated in other types of systems. However, this study does not dismiss another key benefit of 3D main memory, which is the capability to greatly increase the main memory bus width by adding additional TSVs to the memory bus. The benefits of increasing the main memory bus width with 3D memory technology are discussed in Chapter 6.

5.3 Summary

In this chapter, I presented a new, more accurate 3D main memory model that shows the latency reduction going from off-chip metal wires to 3D TSVs is as little as 2.4 ns (or 4.1%) and not the often-quoted 45% to 60% savings. The key reason for this is that the only difference between 2D and 3D main memory is the interconnect latency (off-chip metal wires for 2D and TSVs for 3D) and our circuit simulations show the latency difference between these two technologies to be very small. This is a significant finding because it significantly diminishes one of the key benefits of 3D main memory; significantly faster access to main memory.

As noted in Chapter 1, DSPs traditionally have small caches and with signal processing applications having larger working sets this can lead to high L2 miss rates while traditional 2D main memory has become relatively slower compared to logic (the Memory Wall problem). It was believed that the significant reduction in main memory access latency from 3D main memory would directly address this by allowing caches to quickly fill their lines after a cache miss, but unfortunately that is not the case. Upon re-evaluating the performance benefits of this new 3D main memory latency model, I found that my baseline DSP running the six multimedia

benchmarks showed very little performance improvement. These results indicated that another benefit must be leveraged for 3D main memory to be useful in modern DSPs.

Fortunately, another key benefit of 3D main memory is the capability to significantly increase the main memory bus width through the use of TSVs. Since TSVs take up very little area and do not contribute to the external chip pin count [33], designers can easily increase the main memory bus width by adding more TSVs. With this in mind, Chapter 6 explores the benefits and limitations of increasing the main memory bus width on DSPs with 3D-stacked DRAM and Wide I/O.

My dissertation work described in this chapter was published in the Proceedings of the 2011 International Conference on Embedded Computer Systems (SAMOS) [9][2] and the Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC) [8].

[2] My SAMOS paper evaluated the performance impact of 3D DRAM on DSPs and was done prior to the latency findings outlined in this chapter. The results labeled "Baseline (3D)" are from that paper.

6 Evaluating 3D Main Memory with Wide I/O

This chapter takes the conclusions from the previous chapter (i.e. 3D main memory does not lower main memory latencies by a large amount) and analyzes another key benefit of 3D main memory, which is the ability to increase the main memory bus width with additional TSVs. As noted in the Section 5.3, since TSVs take up very little area, designers can significantly increase the main memory bandwidth (i.e., I/O bus width) without being limited by the package pin constraints. In this chapter, I present a standardized 3D main memory organization called 3D main memory with Wide I/O [74] that takes advantage of TSVs to greatly increase the main memory bus width while also reducing main memory latencies by 9.4 ns (or 15.9%) through main memory architecture optimizations[3].

I also show in this chapter that 3D main memory with Wide I/O can increase the geometric mean performance of a C67x-like DSP by 9.7% (and up to 23.3% on some benchmarks). However, this 3D memory organization can increase average DSP energy consumption by 2.6% (and up to 8.9%), which may be unacceptable for embedded processors found in mobile devices that have stringent energy requirements. Based on these findings, I conclude that if embedded DSPs are to use 3D main memory with Wide I/O, a solution should be found to lower the energy consumption while maintaining the performance benefits.

6.1 3D Main Memory with Wide I/O

Figure 6.1 gives a breakdown of the timing for a low power DDR2 DRAM. For the traditional 2D main memory with the default (i.e. 64-bit) I/O bus, data from the DRAM travels down global and inter-bank datalines, through a 4:1 serializer and then through a serialized dataline. The data

[3] The latency reduction opportunities of 3D main memory with Wide I/O were analyzed in collaboration with Younghoon Son and Professor Jung Ho Ahn at Seoul National University in South Korea.

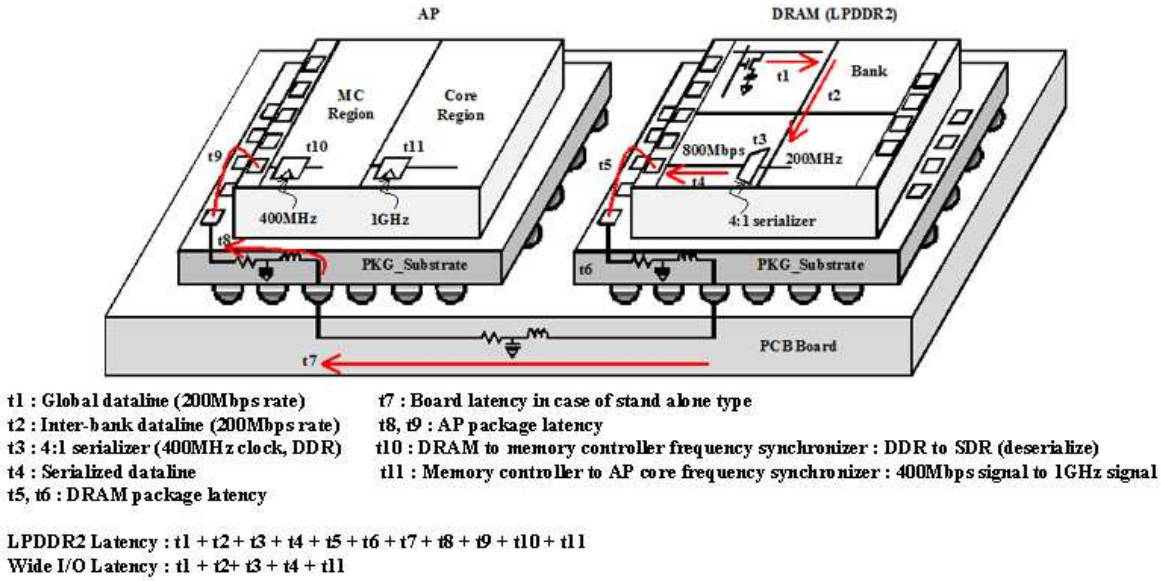

Figure 6.1: A timing breakdown of 2D Low Power DDR2 DRAM.

must then travel through the DRAM package, printed circuit board (PCB), and active package (AP) before arriving at the memory controller (MC). At the MC interface, the data is synchronized with the MC and then synchronized with the processor frequency. Based on the previous chapter, going to 3D integration technology and using TSVs instead of metal interconnects allows the removal of the timing components labeled *t5* through *t9*. Based on this finding, this latency savings was only 1.2 ns (or 2.4 ns roundtrip), which corresponds to 4.1% of the total latency (59 ns).

However, current Wide I/O 3D main memories have two additional reorganizations: 1) removing the DRAM to memory controller synchronizers and 2) placing the address, command, and data pads closer to the DRAM banks. For the first latency savings, DRAM designers remove the component labeled with time *t10*. This component is responsible for both synchronization of the DRAM to the memory controller and rate conversion. The synchronizer also compensates for jitter. Further, since the data transfer rate per pin of Wide I/O 3D main memory is lower (~200 Mbps) than that of low-power DDR2 (~800 Mbps), the synchronizer design for the 3D main

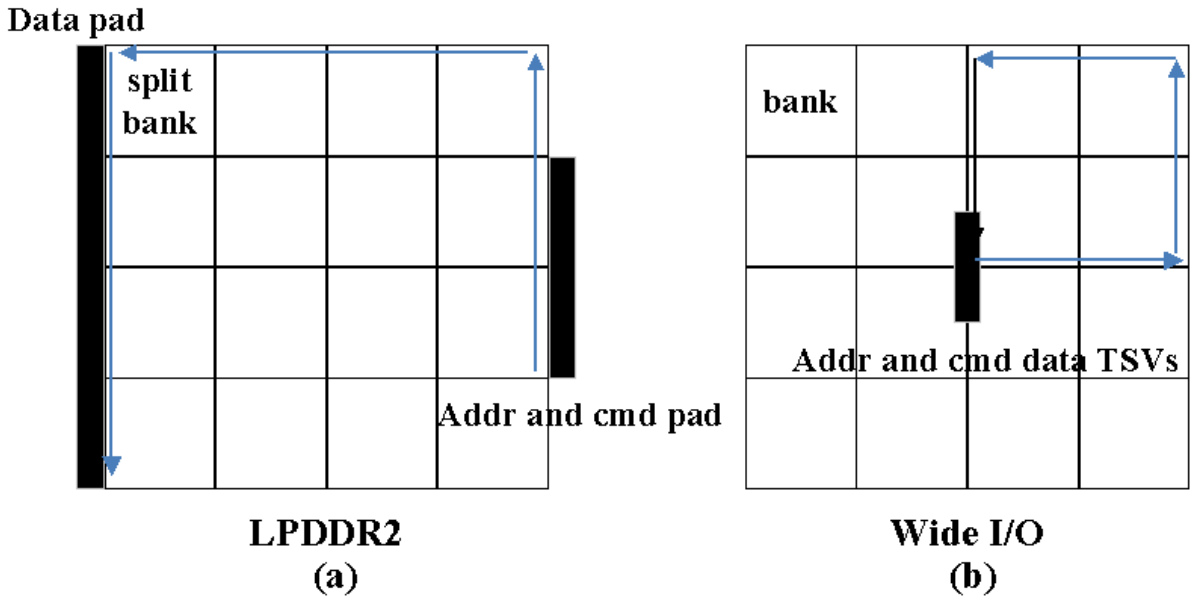

Figure 6.2: Address, command, and data pad placement.

memory with Wide I/O can be simpler, leading to a latency savings. We modified CACTI to model the component that adjusts the difference in transfer rates between an internal and external datapaths and found that it would take one cycle. For a low-power DDR2 DRAM running at 400 MHz, this is 2.5 ns.

The second latency savings for 3D main memory with Wide I/O comes from placing the address, command, and data pads closer to the DRAM banks. Figure 6.2 (a) shows the organization of a traditional low-power DDR2 main memory with 16 DRAM banks. This figure is based on die photos. As can be seen in the figure, a DRAM request coming from the address and command pads on the right side of the figure must traverse both up and across the entire DRAM module before reaching the data pad. Figure 6.2 (b) illustrates the organization for Wide I/O 3D DRAM architectures with TSV interconnects. In this main memory organization, placing the TSVs for the address, command, and data near the center of the DRAM banks translates into each main memory request traveling a shorter total distance. The wire delay was modeled using

the SPICE circuit simulator [75] and the 22 nm low power predictive technology model (PTM) [76] and found that the latency savings was 4.5 ns.

When the latency savings from the previous chapter (2.4 ns) is added to the latency savings of the two DRAM reorganizations described above (2.5 ns and 4.5 ns), the total latency savings is 9.4 ns (2.4 + 2.5 + 4.5), which corresponds to 15.9% for our baseline DRAM (9.4 ns / 59 ns).

6.2 Evaluating 3D Main Memory with Wide I/O on DSPs

In this section, I first detail my experimental methodology and then present a performance analysis of 3D main memory with Wide I/O on an embedded DSP similar to a Texas Instrument C67x. I then present a performance analysis of 3D main memory with Wide I/O where the bus width is increased up to 2,048 (or 4,096) bits, which allows the memory controller to use adjacent cache line prefetching [77] to bring in two (or four) cache lines of data into the L2 cache with a single transfer. This should increase the performance of multimedia applications that traditionally exhibit a high level of spatial locality. For my dissertation work, I did not examine adjacent cache line prefetching without the wider main memory bus (i.e. prefetching one or three 1,024-bit cache lines with a 1,024-bit bus). This should be examined in future work. All wider bus width simulations included adjacent cache line prefetching. Lastly, I analyze the DSP energy consumption of these 3D main memory organizations.

6.2.1 Methodology

I began by simulating the baseline C67x-like DSP detailed in Table 4.1 with traditional 2D main memory and the 3D main memory with Wide I/O described in Section 6.1. Since 3D main memory with Wide I/O is implemented with a wider I/O bus, these wide I/O simulations were done with a main memory bus width of 1,024 bits. 1,024 bits was chosen because, as described

in Section 2.3, during one main memory transaction the memory controller requests the entire L2 cache line to be filled. In a traditional 2D organization the memory controller sends additional column strobes, which result in multiple main memory transactions that eventually fill an entire line in the L2 cache. However, for 3D main memory with Wide I/O, the additional TSVs are used to increase the main memory bus width. Instead of sending multiple 64-bit transactions, the main memory's row buffer can send the entire L2 cache line (1,024 bits in our configuration) in one transaction. These simulations were done to demonstrate the performance benefits of 3D main memory with Wide I/O

Next, I simulated the baseline DSP with Wide I/O 3D DRAM, but increased main memory bus width to 2,048 bits and used adjacent cache line prefetching to fetch an additional 1,024-bit L2 cache line with the additional main memory data lines. This would allow the memory controller to fill two L2 cache lines from one main memory request. Since many multimedia applications have linear access patterns that exhibit high spatial locality, prefetching the next L2 cache line should increase overall DSP performance since the next data to be processed is already in the cache.

Lastly, I again increased the main memory bus width from 2,048 bits to 4,096 bits and again used adjacent cache line prefetching to fetch a total of four 1,024-bit L2 cache lines with the additional main memory data lines. This would allow the memory controller to fill four L2 cache lines from one main memory request. These two configurations were simulated to quantify the performance and energy impact of increasing the main memory bus width and using adjacent cache line prefetching. My simulation parameters are summarized in Table 6.1.

Table 6.1: 3D Main Memory with Wide I/O Simulation Parameters.

Parameter	**Setting**
Frequency	1 GHz
L1 data cache	32-KB, 2-way, 512-bit lines, 1 cycle
L1 instruction cache	32-KB, 1-way, 256-bit lines, 1 cycle
Unified L2 cache	256-KB, 4-way, 1,024-bit lines, 7 cycles
# of ALUs	6
# of multiply units	2
# of memory units	1
Main memory size	512-MB
Main Memory bus widths	64 bits, 1,024 bits, 2,048 bits, 4,096 bits
2D main memory latency	38 to 40 ns
3D main memory with Wide I/O Latency	29 to 31 ns

6.2.2 Results – Performance

Figure 6.3 shows performance results of the six benchmarks for four different main memory and bus width configurations. The second bar per benchmark in Figure 6.3 shows the performance benefits of Wide I/O 3D main memory described in Section 6.1 with all results normalized to the baseline configuration with 2D main memory (the leftmost bar). Compared to the baseline configuration with 2D main memory, the benchmarks achieve a geometric mean speedup of 3.8%. The 482.sphinx, disparity, and mser benchmarks show the largest improvements (5.9%, 8.8%, and 3.4% respectively), which can be attributed to their higher L2 miss rates (39.0%, 44.8%, and 52.2% respectively) and high number of main memory accesses. Although the 15.9% latency reduction is not as high as the often quoted 45% to 60%, it is a large enough latency reduction to improve the performance of each main memory transaction and therefore, the three

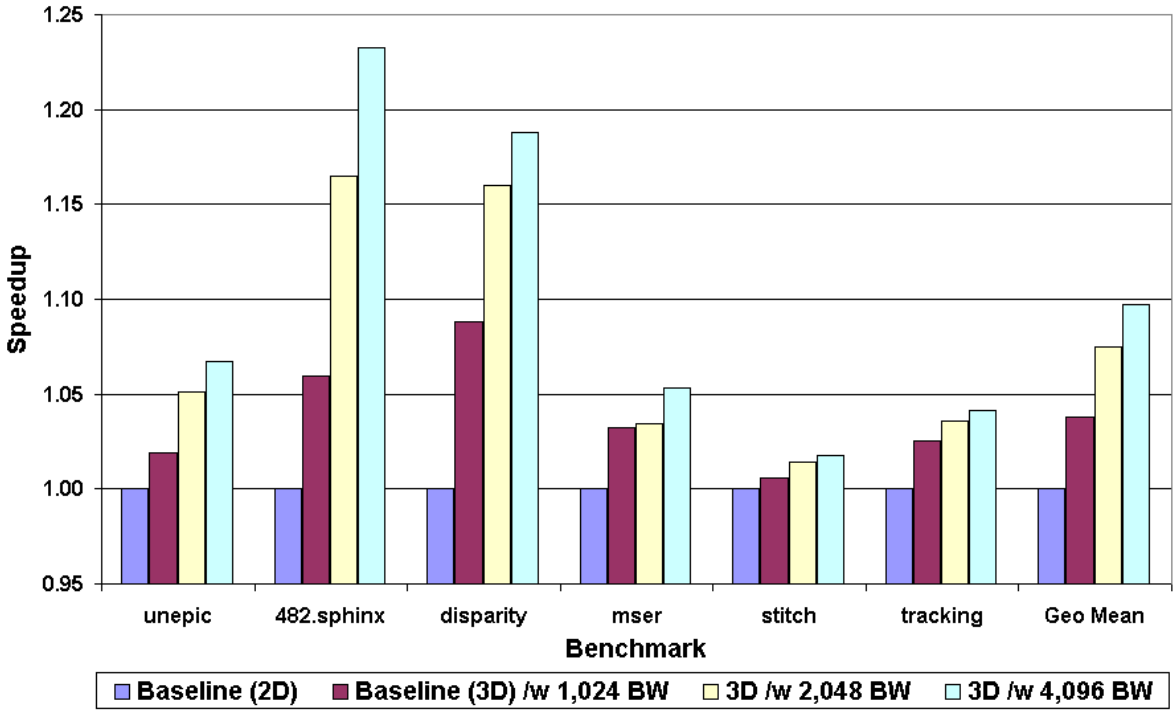

Figure 6.3: Performance results of the six benchmarks on four different main memory and bus width configurations.

applications with the most main memory transactions benefited significantly. The remaining benchmarks demonstrate speedups of 0.6% to 2.5%.

The third bar per benchmark in Figure 6.3 shows the performance benefits of Wide I/O 3D main memory, increasing the main memory bus width to 2,048 bits, and adjacent cache line prefetching of one additional line with all results normalized to the baseline configuration with 2D main memory. Compared to the baseline with 2D main memory and the default main memory bus width, the benchmarks achieve a geometric mean speedup of 7.5%. The unepic, 482.sphinx, and disparity benchmarks demonstrate the largest improvements (5.1%, 16.5%, and 16.0%, respectively), which can be attributed to their L2 and L1 miss rates decreasing. The remaining benchmarks show speedups of 1.4% to 3.6%. The average L2 miss rate (using the geometric mean) of the six benchmarks went from 22.4% (the baseline) down to 16.7%. Since each main memory transaction is now bringing in two adjacent L2 cache lines instead of one,

applications that exhibit a high level of spatial locality will hit more often in the L2 cache leading to fewer main memory transactions and increased performance.

The rightmost bar per benchmark in Figure 6.3 shows the performance benefits of 3D main memory with Wide I/O, increasing the main memory bus width to 4,096 bits, and adjacent cache line prefetching of three additional lines with all results normalized to the baseline configuration with 2D main memory. Compared to the baseline configuration with 2D main memory and the default bus width, the six benchmarks achieve a geometric mean speedup of 9.7%. The unepic, 482.sphinx, disparity, and mser benchmarks show the largest improvements (6.7%, 23.3%, 18.8%, and 5.3%, respectively), which again can be attributed to a decrease in the L2 miss rate. The remaining benchmarks demonstrate speedups ranging from 1.8% to 4.1%. The average L2 miss rate (using the geometric mean) of the six benchmarks went from 22.4% (the baseline) down to 13.0%. Similar to the previous paragraph, since each main memory transaction is now filling four adjacent L2 cache lines instead of one, applications that exhibit a high level of spatial locality will hit more often in the L2 leading to a lower L2 miss rate and higher performance.

These results show that 3D main memory with Wide I/O and prefetching adjacent cache lines can increase DSP performance by an average of 9.7% and up to 23.3%. Although a portion of the speedup for each Wide I/O configurations can be attributed to the reduced main memory latency from the main memory organization described Section 6.1, the majority comes from the increased main memory bus width and adjacent L2 cache line prefetching. This indicates that the benchmarks show a high amount of spatial locality and can take advantage of the increased memory bus width between the L2 and main memory. This high level of spatial locality is not surprising since all six of our benchmarks are streaming multimedia applications, which are commonly run on mobile devices and have fairly linear memory access patterns.

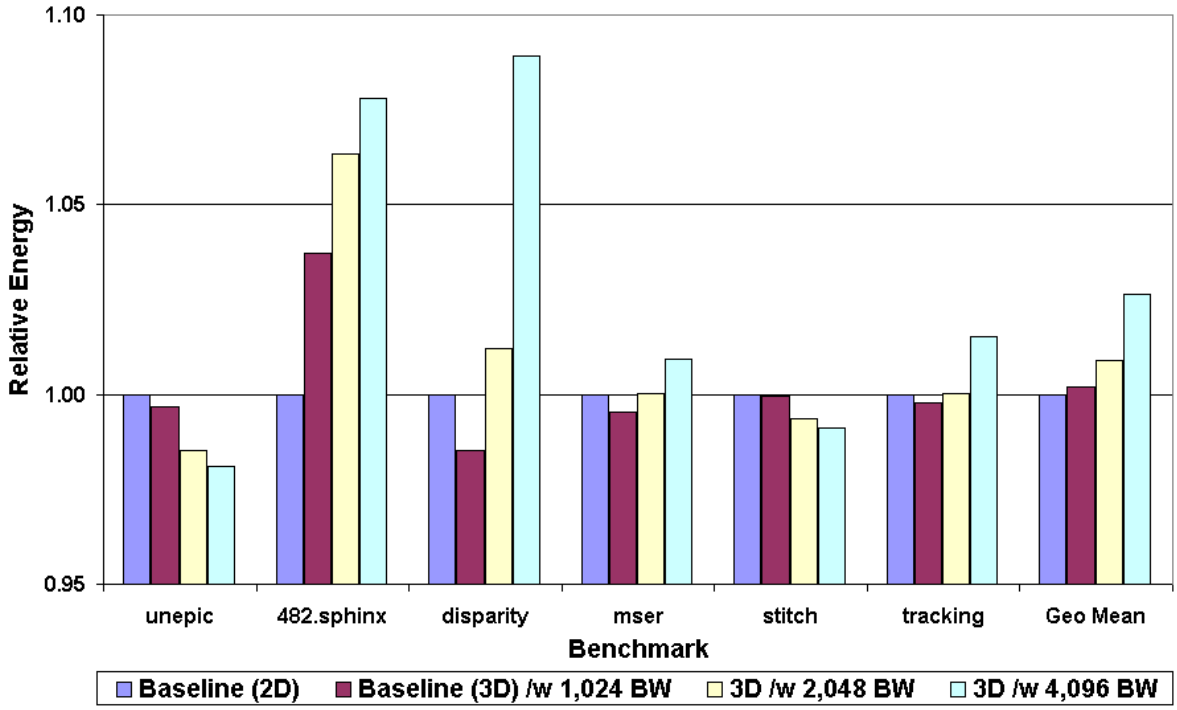

Figure 6.4: Energy results of the six benchmarks on four different main memory and bus width configurations.

6.2.3 Results – Energy Consumption

Figure 6.4 shows the relative energy results for the six benchmarks on the four main memory and bus width configurations with all results normalized to the baseline with 2D main memory (the leftmost bar per benchmark). The second bar per benchmark in Figure 6.4 shows that going to 3D main memory with Wide I/O has a slight increase in DSP energy consumption. We observe, on average, a 0.2% increase in energy with many applications actually seeing a small reduction in energy consumption. However, 482.sphinx yields a 3.7% increase in DSP energy consumption, which skews the average and is explained below.

The decrease in DSP energy consumption for five of the benchmarks can be partially attributed to the 3D main memory's use of TSVs, which consume 11.2x less I/O energy than off-

chip metal interconnects [29]. However, all five benchmarks actually show a slight increase in I/O energy since the main memory bus width was increased beyond the 11.2x energy savings (1,024 bits was a 16x increase). Yet, the performance improvements of 3D main memory with Wide I/O yields energy decreases in the decoding unit, integer unit, etc. Since I/O energy is only a fraction of total DSP energy consumption, an increase in one area (I/O energy) does not always yield a net increase in energy as long as other components see energy improvements.

However, as 482.sphinx demonstrates, increasing the main memory bus width beyond 11.2x (1,024 bits was a 16x increase) can lead to not only an increase in I/O energy, but also an increase in overall DSP energy consumption. Compared to the other five benchmarks, 482.sphinx had slightly more main memory transactions per instruction meaning this benchmark generated more main memory transactions throughout its execution. This means 482.sphinx's I/O energy represents a higher portion of the total DSP energy. Although all six benchmarks had an increase in I/O energy because they increased the main memory bus width beyond 11.2x, 482.sphinx's increase was larger, which resulted in a net increase in DSP energy consumption. As will be shown in the next set of results, continuing to increase the main memory bus width and increasing I/O energy can eventually lead to energy inefficiency for not only 482.sphinx, but most of the benchmarks.

The third bar per benchmark in Figure 6.4 shows that increasing main memory bus width to 2,048 bits and adjacent cache line prefetching of one additional line results in an increase in energy consumption when compared to the baseline with 2D main memory. The configuration with the increased bus width and adjacent cache line prefetching consumes, on average, 0.9% more energy than the baseline configuration with 2D main memory. This can be attributed to the

main memory bus width being increased by 32x and adjacent cache line prefetching evicting valid L2 cache entries leading to an increase in energy consumed by the L2 cache.

Although 3D main memory provides an 11.2x reduction in I/O energy per bit transfer, increasing the main memory bus width by 32x means the I/Os will consume more energy than the default 64-bit bus. However this increase in I/O energy only partially explains the increase in energy. The processor now brings in 2,048 bits (two L2 cache lines) instead of 64 bits per transaction due to the wider I/O bus width and adjacent cache line prefetching. Many times, bringing in a larger amount of data and replacing the L2 cache lines leads to evicting a large number of valid entries. This leads to I/O energy inefficiency where the L2 cache needs to send additional main memory requests to undo the evictions. Although 3D main memory provides an energy decrease per bit, if the memory transactions are inefficient, there is energy wasted on a non-useful transaction. For example, if the main memory bus width was the default setting this would lead to fewer L2 cache lines being evicted. Many times, the lower bus width setting may be better for a particular program phase that does not have many main memory accesses that exhibit spatial locality. This was the case for 482.sphinx whose L2 cache energy consumption increased by a factor of two. However, since the L2 cache and main memory are only a fraction of the total energy consumed by the DSP, the total energy increase is still relatively small. 482.sphinx and disparity show the largest increases in relative energy (6.3% and 1.2%, respectively).

Surprisingly two benchmarks (unepic and stitch) demonstrate a decrease in energy consumption (1.5% and 0.6%, respectively). Although both benchmark's I/O energy consumption was higher (as described above), their L2 cache energy consumption decreased

significantly (34.4% and 36.0%, respectively), which overall leads to the DSP consuming less energy.

The rightmost bar per benchmark in Figure 6.4 shows the relative energy results of the six benchmarks with Wide I/O 3D main memory, the main memory bus width increased to 4,096 bits, and adjacent cache line prefetching of three additional lines. The configuration with the increased bus width and adjacent cache line prefetching consumes, on average, 2.6% more energy than the baseline configuration with 2D main memory. This again can be attributed to the main memory bus width being increased beyond 11x (64x in this case) and this leading to some of the main memory transactions being energy inefficient and evicting valid L2 cache entries. 482.sphinx and disparity demonstrate the largest increases in relative energy (7.8% and 8.9%, respectively).

The unepic and stitch benchmarks again achieve a decrease in energy consumption (1.9% and 0.9%, respectively), which again can be attributed to a decrease in both benchmark's L2 cache energy consumption (49.5% and 51.8%, respectively). Four of the six benchmarks (excluding 482.sphinx and disparity) show a decrease in L2 cache energy consumption, but for mser and tracking, the increase in I/O energy consumption due to the wider bus width canceled this out.

Although the current implementation of 3D main memory with Wide I/O can improve performance by prefetching adjacent L2 cache lines in advance, these results show that it can sometimes lead to I/O inefficiency due to incorrectly evicting correct L2 entries. These results lead to an interesting dilemma where designers can either choose 3D main memory with Wide I/O to increase DSP performance and increase energy consumption or choose slower 2D main

memory since it does not impact DSP energy consumption. This particular scenario motivated me towards designing my dynamic bandwidth scaling algorithms, which is detailed in Chapter 7.

6.3 Summary

In this chapter, I presented a model for 3D main memory called 3D main memory with Wide I/O. 3D main memory with Wide I/O take advantage of TSVs consuming less area than off-chip metal interconnects and uses this benefit to greatly increase the bus width between main memory and the L2 cache. 3D main memory with Wide I/O also offers two main memory reorganizations: 1) removing the DRAM to memory controller synchronizer and 2) placing the address, command, and data pad TSVs closer to the DRAM bank edges. These two changes can yield a 7.0 ns savings in main memory latency, which when combined with the 2.4 ns savings of going from off-chip metal interconnects to TSVs described in Chapter 5, leads to a total main memory latency reduction of 9.4 ns (or 15.9%). Although not as high as the original latency saving claim for 3D main memory, when this 9.4 ns latency savings is combined with increasing the main memory bus width and adjacent cache line prefetching, 3D main memory with Wide I/O can lead to a 9.7% average performance improvement on the six multimedia benchmarks running on a C67x-like DSP (and up to 23.3% on some benchmarks).

However, although 3D main memory with Wide I/O can significantly improve performance, I also show that increasing the main memory bus width beyond a factor of 11.2x can also increase I/O energy. Although many times, the increased main memory bus width can improve performance during memory intense phases, it can also evict valid L2 entries, leading to I/O inefficiency. Upon evaluation, I found that increasing the main memory bus width to 4,096 bits can lead to an increase in DSP energy consumption by an average of 2.6% and up to 8.9% for some applications. Although DSPs have growing memory requirements that 3D main memory

with Wide I/O addresses, this increase in energy consumption may be unacceptable since many of these DSPs are found on devices such as smart phones that require low energy usage.

These findings motivated me towards designing a solution that would allow DSPs to take advantage of the performance benefits of 3D main memory with Wide I/O while also minimizing the I/O energy to keep the DSP energy consumption closer to the baseline. In Chapter 7, I detail my dynamic bandwidth scaling algorithms, which solves this problem by dynamically increasing the main memory bus width during memory-intense program phases to improve DSP performance, but also decreases the bus width when it is no longer needed. With these algorithms, the processor is no longer in the increased the bus width settings with cache line prefetching for the entire execution, which leads to more efficient usage of the wider I/O bus width.

My dissertation work described in this chapter has been accepted for publication at the IEEE/ACM 2013 International Conference on Computer-Aided Design (ICCAD) [10].

7 Dynamic Bandwidth Scaling Algorithms for DSPs

In this chapter, I first demonstrate that an application can have very different performance characteristics based on its program phases and the main memory bus width setting it is in. I then show that this observation could be harnessed to address the problems highlighted in the previous chapter (i.e. although the current implementation of 3D main memory with Wide I/O can significantly increase DSP performance, it also increases DSP energy consumption). This leads me to propose novel algorithms for dynamic bandwidth scaling. My dynamic bandwidth scaling algorithms can dynamically increase and decrease the main memory bus width of the DSP based on the program phases of the application. In memory-intense phases, more TSVs are activated and the main memory bus width is increased to improve performance while during compute-intense phases with little memory traffic, the bus width is decreased to reduce I/O energy consumption. This is in contrast to the current methodology where processors with 3D main memories and Wide I/O stay in the higher bus width setting for the entire execution of a program, which as demonstrated in Chapter 6, can waste I/O energy.

Later in this chapter, I propose three different algorithms to dynamically determine the main memory bus width and show they can improve average performance by 6.6% while increasing DSP energy consumption by only 0.5% (compared to the 2.6% quoted in Chapter 6). Based on these results, I conclude that my dynamic bandwidth scaling algorithms offer a solution to the growing memory requirements of modern signal processing applications while still meeting the strict energy requirements of mobile devices such as smart phones.

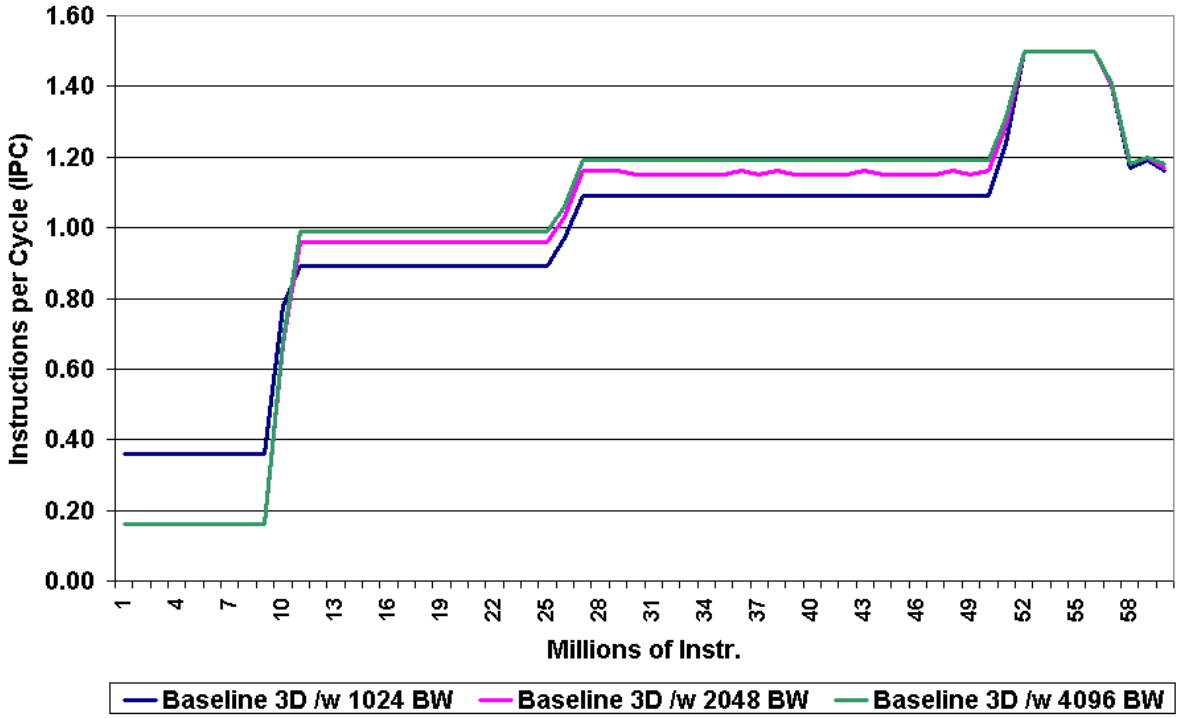

Figure 7.1: IPC behavior of mser (face detection) at different main memory bus widths (BW).

7.1 Motivation

As demonstrated in Chapter 6, Wide I/O 3D DRAM can show large performance improvements on some of our benchmarks (up to 23.3%). However, although the TSVs used in 3D DRAM consume 11.2x less I/O energy than off-chip metal interconnects, this energy savings disappears when the main memory bus width is increased by a large factor. Since the main memory bus width was increased up to 4,096 bits, running this bus width configurations yielded an increase in DSP energy consumption by an average of 2.6% (and as high as 8.9%). For embedded DSPs that have stringent energy constraints, this energy increase can lead to lower battery life on mobile computing devices.

To illustrate the behavior described in the previous chapter regarding inefficient main memory usage at high bus width settings, Figure 7.1 shows the instructions per cycle (IPC) of a

60 million instruction segment of the mser (face detection) benchmark at three main memory bus width settings (1,024, 2,048, and 4,096 bits). This segment was tens of millions of instructions into the execution with the entire benchmark taking over two billion cycles to complete. This segment was specifically chosen since it shows the different program phases of a typical multimedia application. For the first 10 million instructions, the 1,024 bus width setting has the best performance (0.36 compared to 0.18 for the maximum bus width setting) indicating that during this program segment increasing the main memory bus width actually degrades performance while also consuming more energy.

However, for the next 40 million instructions, both higher main memory bus width settings have better performance with the maximum bus width setting achieving the best performance (an average IPC of 1.19 compared to 1.09 for the baseline) indicating that the face detection benchmark is in a memory-intense phase requiring a large amount of main memory bandwidth. During this 40 million instruction segment, increasing the main memory bus width would improve the performance of this application.

Finally, for the last 10 million instructions all three bandwidth settings have the same performance. This indicates that the benchmark is in a compute-intense phase with very few main memory transactions and again, the lowest main memory bus width setting would be the most energy efficient. This program segment illustrates that being in the maximum main memory bus width configuration does not always translate into better performance. Based on the energy results in the previous chapter, this can lead to energy inefficiency since the higher bus width settings will send larger amounts of data to fill more L2 lines, which evicts more L2 entries. These observations led me to design dynamic bandwidth scaling algorithms, which are described in the next section.

7.2 Dynamic Bandwidth Scaling Algorithms

As described at the beginning of this chapter, my dynamic bandwidth scaling algorithms can dynamically increase the main memory bus width and fetch adjacent L2 cache lines during memory intensive phases of a program. As shown in the previous chapter, since many modern signal processing applications are multimedia-based and have predictable, linear access patterns, preemptively fetching additional L2 cache lines can reduce the number of main memory accesses and greatly increase DSP performance. However, unlike the static main memory bus width settings analyzed in Chapter 6, my dynamic bandwidth scaling algorithms can also decrease the main memory bus width when it is no longer needed so as to more efficiently use the increased main memory bandwidth and decrease I/O energy consumption. Unlike the Wide I/O configurations in Chapter 6, my dynamic bandwidth scaling algorithms are not in the maximum bus width configurations for the entire execution, which leads to better I/O energy efficiency while still improving performance.

My dynamic bandwidth scaling algorithms can be implemented with the use of memory controller signals and adding multiplexers between the interbank datalines and the buffers connected to the I/O pads. The multiplexer select bits are controlled by the memory controller and operating system, which would run one of the scaling algorithms described in the next section. Figure 7.2 shows a potential implementation for a simplified configuration, where each memory bank outputs at most four bits.

For this example, to scale the bus width down by a factor of four, the three leftmost I/O pads would be disabled and using the multiplexer select bits on the rightmost 4:1 multiplexer, the correct bit would be selected and go through to the rightmost I/O pad. Similarly, to scale down the bandwidth by a factor of two, the two leftmost I/O pads would be disabled and the 2:1 and

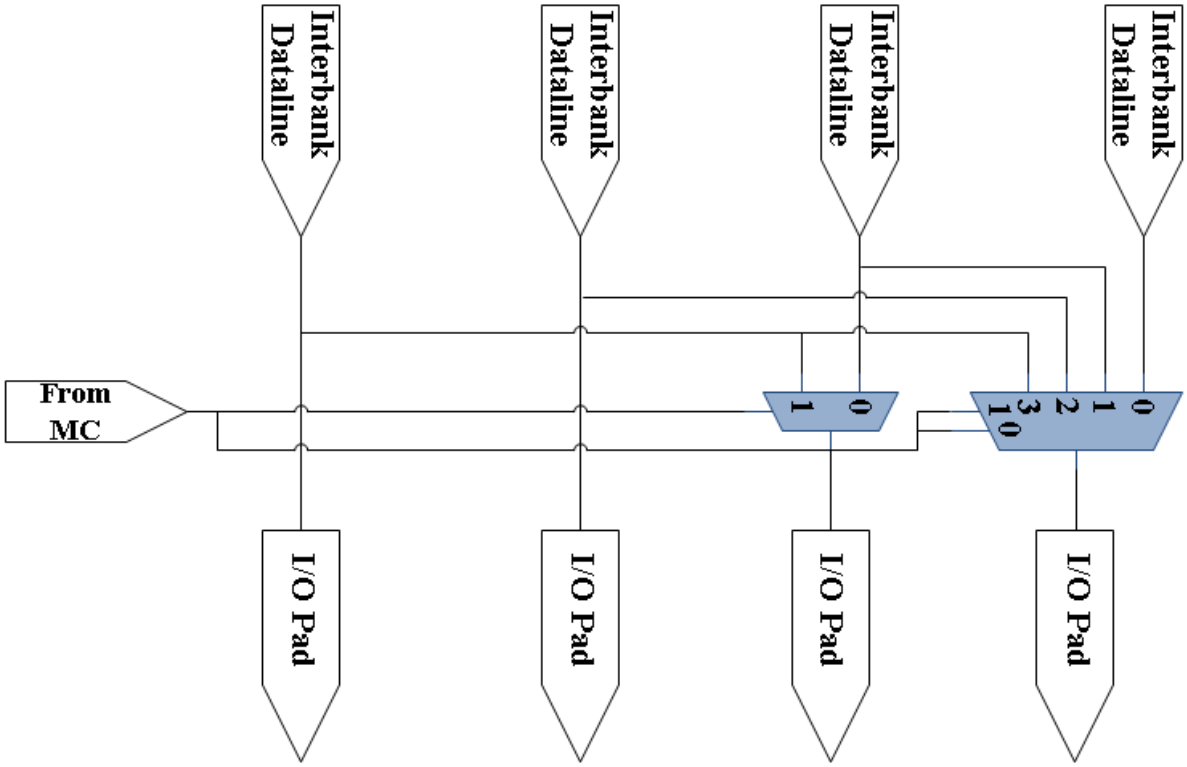

Figure 7.2: An example of the dynamic bandwidth scaling hardware.

4:1 multiplexers on the right would be used to select the correct two bits. With this configuration the correct data should come out regardless of what bus width configuration is chosen since all four bits are available to the 4:1 multiplexer and two of the four bits are available to the 2:1 multiplexer. Similarly, all four I/O pads can be enabled to provide the maximum main memory bandwidth. The 2:1 and 4:1 multiplexers were simulated in Synopsys [78] and for a 4,096-bit bus width they were found to consume 1.9 nJ and 3.8 nJ, respectively, which is not a large increase relative to the total energy consumption of our benchmarks.

In the following sections, I first describe the methodology used in designing my dynamic bandwidth scaling algorithms. I then describe three different dynamic bandwidth scaling algorithms and lastly I analyze their performance and energy results.

7.2.1 Algorithm Design Methodology

I modified the Trimaran simulator code to profile the six benchmarks and generated two sets of profile data, which were taken every one million instructions: 1) the average instructions per cycle (IPC) and 2) the percentage of consecutive L2 misses that map to adjacent L2 cache lines. The IPC data allowed me to look for execution phases as well as the upward or downward trends in performance. The percentage of consecutive L2 misses that map to adjacent L2 cache lines tracked whether the addresses of the current L2 miss and previous L2 miss mapped to adjacent cache lines. This percentage included both directions (i.e. increasing and decreasing addresses). This statistic implied that the program was in a streaming part of the code and increasing the main memory bus width would increase performance by allowing the program to fetch an additional, adjacent L2 cache line and reduce the L2 miss rate.

The percentage of consecutive L2 misses that map to an adjacent L2 cache line could be tracked by adding a comparator that compares the current and previous main memory addresses and then increments a performance counter if the two addresses are adjacent to each other. The IPC data could be obtained by storing the execution cycles in a performance counter and then resetting the counter after every one million instructions. Lastly, as noted in Section 7.2, my dynamic bandwidth scaling algorithms would be executed by the memory controller and could be programmed into the operating system.

After collecting all of the profile data, I analyzed the two statistics and found that the benchmarks had program phases where IPC increases or decreases dramatically based on which static bus width configuration (1,024, 2048, or 4,095 bits) they were in, which was illustrated in Section 7.1. I also parsed the percentage of consecutive adjacent L2 line misses and analyzed the memory access patterns for each benchmark looking for program segments with a high amount

of consecutive L2 misses that map to an adjacent L2 cache line. After identifying these program segments, I tested a variety of percentages from 1.0% to 50.0% to see at what percentage increasing the main memory bus width would benefit the application. I found that when the benchmarks had a consecutive adjacent L2 line miss percentage greater than 3.0%, they usually benefited from having a larger main memory bus width. Although the applications had many program segments where the consecutive adjacent L2 line miss percentage was greater than 3.0% (some as high as 50.0%), I found that this low threshold still offered opportunities to improve the performance of the DSP. With these findings in mind, I developed three bandwidth scaling algorithms, which are described below.

I chose to have my algorithms make a bus width decision every five million instructions. This granularity was chosen after simulating a variety of other settings such as every one million instructions, ten million instructions, etc. One million was not chosen since it meant the algorithm would change the bus width too frequently and ten million was not chosen because if the algorithm made an incorrect bus width decision, the benchmark would run in that setting for too long.

7.2.2 Consecutive Adjacent L2 Line Miss Bandwidth Scaling Algorithm

Figure 7.3 shows a small portion of the 482.sphinx benchmark's main memory requests with the main memory addresses highlighted on the left in blue. This segment shows that the main memory requests for 482.sphinx have a very linear pattern where the address is continually incremented by offsets of 0x80 (in hexadecimal), where 0x80 is 128 bytes (or 1,024 bits), which is also the L2 line size for our C67x-like DSP.

With this observation in mind, the consecutive adjacent L2 line miss bandwidth scaling algorithm was designed under the assumption that if the consecutive adjacent L2 line miss

Address	RD/WR	Cycle
0x09741000	P_MEM_RD	846939
0x09741080	P_MEM_RD	847084
0x14a3b180	P_MEM_RD	847205
0x14a3b200	P_MEM_RD	848017
0x14a3b280	P_MEM_RD	849873
0x14a3b300	P_MEM_RD	851729
0x14a3b380	P_MEM_RD	853585
0x14a3b400	P_MEM_RD	855441
0x14a3b480	P_MEM_RD	857297
0x14a3b500	P_MEM_RD	859153
0x14a3b580	P_MEM_RD	861009
0x14a3b600	P_MEM_RD	862865
0x14a3b680	P_MEM_RD	864721

Figure 7.3: Example of a main memory request pattern for 482.sphinx (speech recognition).

percentage is greater than or equal to 3.0%, the benchmark may be in a streaming section. If a program is continuously missing adjacent L2 cache lines, fetching them in advance should decrease the L2 miss rate and improve DSP performance. Therefore, this algorithm only uses the percentage of consecutive L2 misses that map to an adjacent L2 cache line to make a main memory bus width decision. Figure 7.4 illustrates the algorithm, which is described below.

The algorithm begins in the lowest main memory bus width setting (1,024 bits). If the percentage of consecutive adjacent L2 line misses exceeds the threshold (3.0%) three or more times in the last five intervals (an interval is one million instructions), the main memory bus width is increased to the next higher setting (2,048 bits or 4,096 bits) for the next five million instructions. If the percentage of consecutive adjacent L2 line misses is less than 3.0% for three or more intervals, the main memory bus width is decreased to the next lower setting (2,048 bits or 1.024 bits) since this profile data suggests that the program is not in a streaming section. If the algorithm reaches the maximum setting and the next bus width decision is to increase the bus

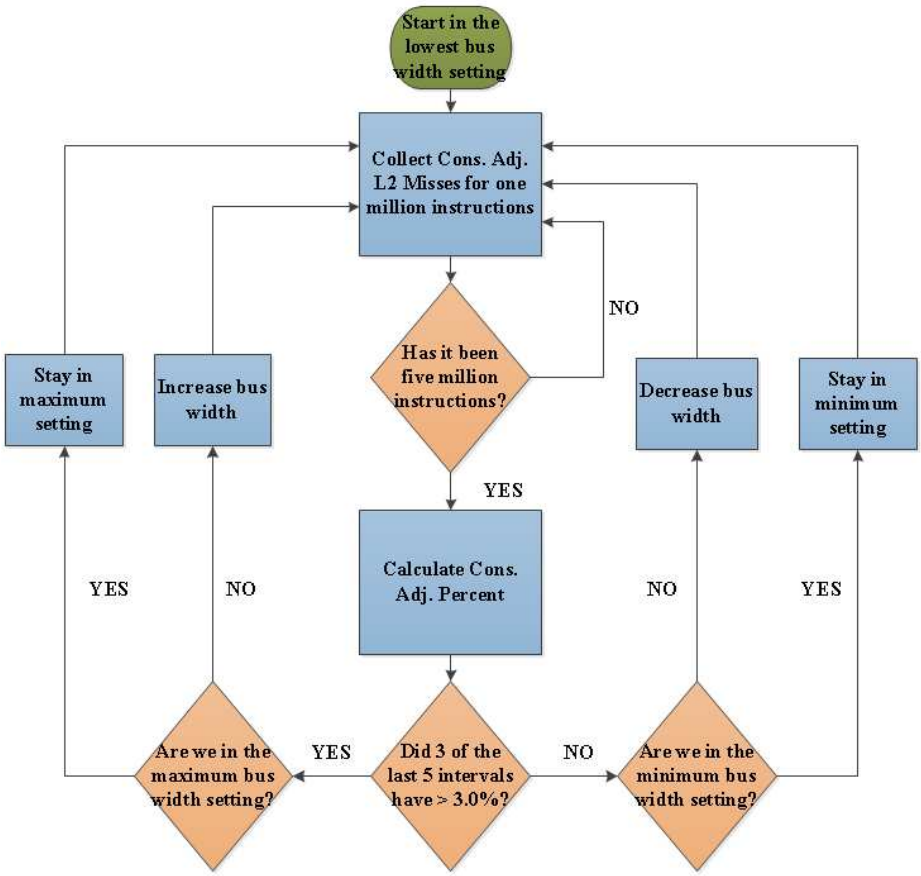

Figure 7.4: Flowchart for the consecutive adjacent L2 line miss algorithm.

width, it remains in the maximum setting. Similarly, if the algorithm reaches the minimum setting and the next decision is to decrease the bus width, it remains in the minimum setting.

The three out every five intervals criteria was also chosen after testing a variety of other configurations such as five out of every ten, etc. However, since I did not test every permutation of this, it is quite possible better criteria exist.

7.2.3 IPC Moving Average Bandwidth Scaling Algorithm

The IPC moving average bandwidth scaling algorithm assumes that if the IPC trend of the last five intervals is downward, the degradation of the benchmark's performance may be due to a lack of main memory bandwidth. It is possible that degradation in IPC is unrelated to the main

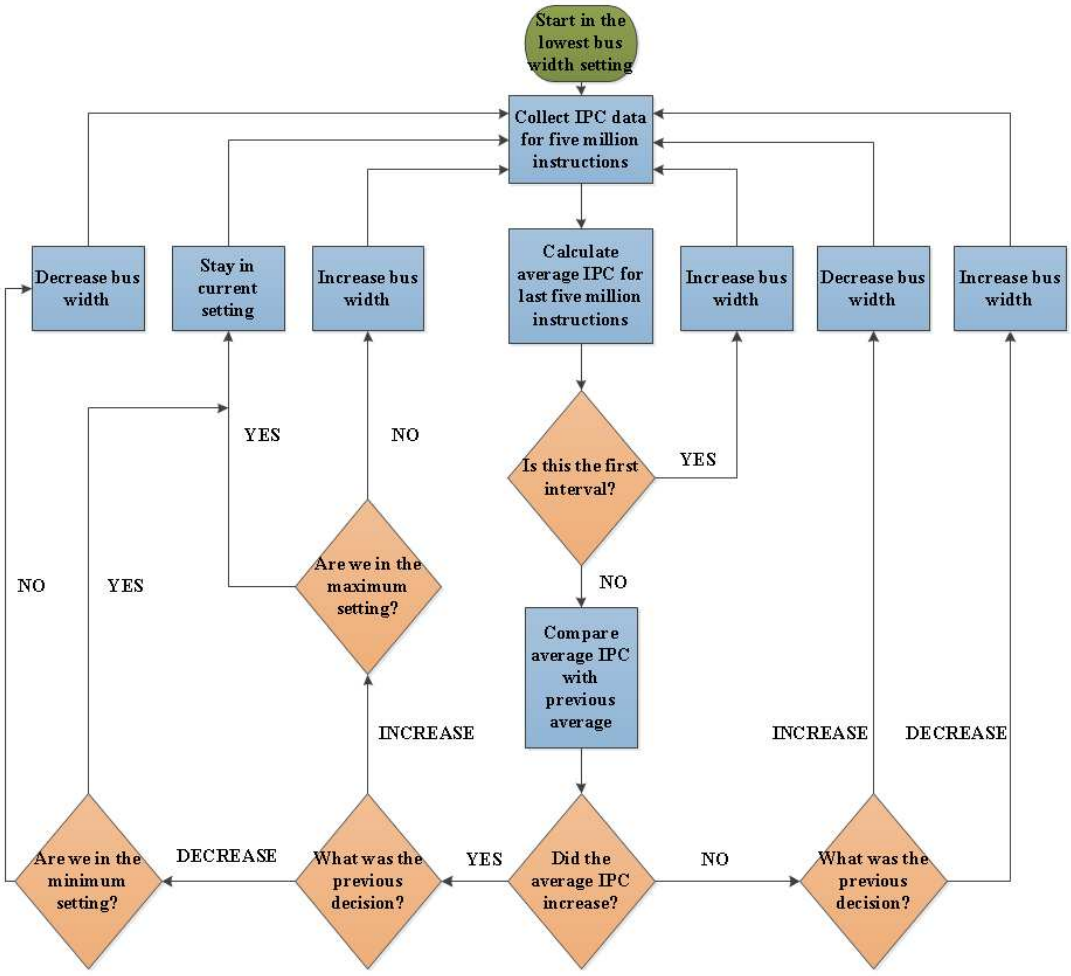

Figure 7.5: Flowchart for the IPC moving average algorithm.

memory bandwidth (e.g. a program phase with many branches). This bandwidth scaling algorithm uses the IPC data of the previous five million instructions to calculate the IPC average before making a bus width decision. Figure 7.5 illustrates the algorithm, which is described below.

The algorithm begins by starting in the lowest main memory bus width setting (1,024 bits) and executing for five million instructions to establish the average IPC for one interval. The bus width is then increased by a factor of two to 2,048 bits for the next five million instructions to establish what impact increasing the bus width has on the IPC trend. If the average IPC

decreased when going from the 1,024-bit setting to the 2,048-bit setting, the main memory bus width is decreased back to the default bus width setting (1,024 bits) since it is possible fetching additional L2 cache lines is the reason for the decrease in performance. If the average IPC in the 2,048-bit setting increased when going from the 1,024-bit setting, the bus width is again increased since the previous increase in bus width may be the reason for the increase in IPC.

In general, if the algorithm's previous decision was to increase the bus width and the IPC trend for the next interval is upward, it increases the bus width again. However, if the algorithm's previous decision was to increase the bus width and the IPC trend for the next interval is downward, it decreases the bus width. Similarly, if the algorithm's previous decision was to decrease the bus width and the IPC trend for the next interval is downward, it increases the bus width. However, if the algorithm's previous decision was to decrease the bus width and the average IPC increases, the algorithm decreases the bus width again.

This is done at each setting, where based on the IPC trend of the last interval and what the previous bus width decision was (increase or decrease in bus width), the algorithm makes a bus width decision. If the maximum setting (4,096 bits) is reached and the average IPC trend is still increasing, the DSP remains in this setting since a higher setting does not exist. If the minimum setting (1,024 bits) is reached and the average IPC trend is still increasing, the DSP remains in this setting.

7.2.4 Combined Bandwidth Scaling Algorithm

The combined bandwidth scaling algorithm is a combination of the above two algorithms. The combined bandwidth scaling algorithm increases the main memory bus width if at least one of following the three criteria is met: 1) the percentage of consecutive adjacent L2 line misses is greater than 3.0% for three of the last five intervals, 2) the previous decision was to increase the

bus width and the average IPC increased, or 3) the previous decision was to decrease the bus width and the average IPC went down. The combined bandwidth scaling algorithm decreases the main memory bus width if at least one of following the three criteria is met: 1) the percentage of adjacent L2 line misses is less than 3.0% for three of the last five intervals, 2) the previous decision was to decrease the bus width and the average IPC increased, or 3) the previous decision was to increase the bus width and the average IPC decreased.

7.2.5 Oracle Scaling Algorithm

An oracle bandwidth scaling algorithm was also implemented to determine how close my algorithms were to the maximum attainable performance. The oracle scaling algorithm takes all the profiling data described in Section 7.2.1 and chooses the main memory bus width configuration that maximizes the IPC at each interval. The performance results of my algorithms were compared to the oracle algorithm's results to determine how effective they were.

7.3 Evaluating Dynamic Bandwidth Scaling Algorithms

In this section, I first detail my experimental methodology and then present a performance analysis of my three dynamic bandwidth scaling algorithms running on an embedded DSP similar to a Texas Instrument C67x. I then present an energy analysis of each of my algorithms and conclude that my dynamic bandwidth scaling algorithms not only improve the performance of the baseline DSP, but do so while consuming less energy than statically scheduling the maximum main memory bus width configuration for the entire execution duration of an application.

7.3.1 Methodology

The three dynamic bandwidth scaling algorithms described above were all implemented and then run on a C67x-like DSP within the Trimaran simulator. My simulation parameters are summarized in Table 7.1.

Table 7.1: Dynamic Bandwidth Scaling Simulation Parameters.

Parameter	**Setting**
Frequency	1 GHz
L1 data cache	32-KB, 2-way, 512-bit lines, 1 cycle
L1 instruction cache	32-KB, 1-way, 256-bit lines, 1 cycle
Unified L2 cache	256-KB, 4-way, 1,024-bit lines, 7 cycles
# of ALUs	6
# of multiply units	2
# of memory units	1
Main memory size	512-MB
Main Memory bus width settings	1,024 bits, 2,048 bits, 4,096 bits
3D main memory with Wide I/O Latency	29 to 31 ns
Bandwidth Scaling Algorithms Tested	Consecutive, Moving Average, Combined

The performance and energy results of each of the three dynamic bandwidth scaling algorithm were then compared to the original baseline DSP with traditional 2D main memory and the oracle bandwidth scaling algorithm. This was done to not only show their improvement over the original 2D baseline, but to also show how close my algorithms were to the oracle bandwidth scaling algorithm's performance.

7.3.2 Results - Performance

Figure 7.6 shows the performance of my three dynamic bandwidth scaling algorithms, a static bus width of 4,096 bits, and the oracle scaling algorithm all with 3D memory and Wide I/O with

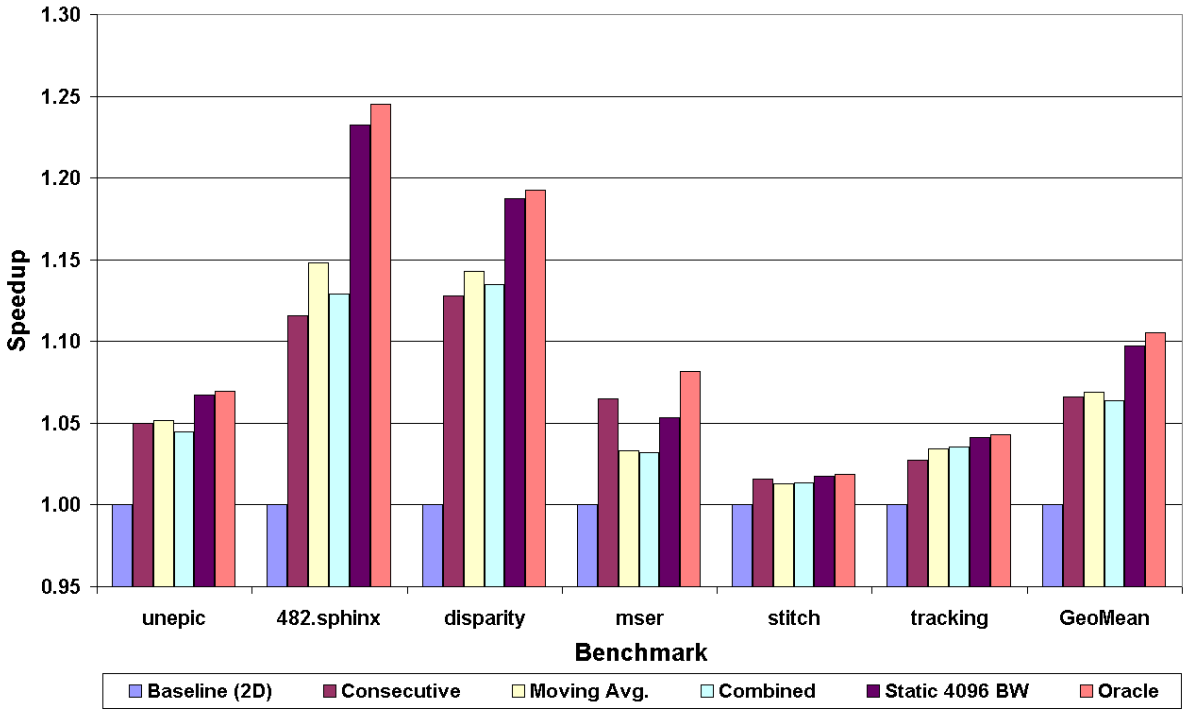

Figure 7.6: Performance results of my three bandwidth scaling algorithms on the six benchmarks.

all results normalized to the baseline processor with 2D main memory and the default bus width (the leftmost bar per benchmark). The rightmost bar per benchmark shows that the oracle algorithm has an average (geometric mean) performance improvement of 10.6% when compared to the baseline with 2D main memory and the default bus width. The 482.sphinx, disparity, and mser benchmarks showed the largest improvements (24.5%, 19.3%, and 8.2%, respectively). The oracle algorithm provides a ceiling for the maximum performance improvement that any bandwidth scaling algorithm can attain.

The second bar per benchmark in Figure 7.6 shows the performance of the consecutive adjacent L2 line miss bandwidth scaling algorithm on the six benchmarks. The consecutive adjacent L2 line miss algorithm achieves an average (geometric mean) speedup of 6.6% when compared to the baseline with 2D main memory and the default bus width. The 482.sphinx,

disparity, and mser benchmarks saw the largest speedups (11.6%, 12.8%, and 6.5%, respectively) since speech recognition, 3D reconstruction, and face recognition are all memory intense streaming applications with high L2 miss rates. The 482.sphinx, disparity, and mser benchmarks originally had L2 miss rates of 39.0%, 44.8%, and 52.2%, respectively.

The third bar per benchmark in Figure 7.6 shows the performance of the IPC moving average bandwidth scaling algorithm on our six benchmarks. The IPC moving average algorithm achieves an average (geometric mean) speedup of 6.9% when compared to the baseline with 2D main memory and the default bus width. The unepic, 482.sphinx, and disparity benchmarks showed the largest speedups (5.2%, 14.8%, and 14.3%, respectively).

The fourth bar per benchmark in Figure 7.6 shows the performance of the combined bandwidth scaling algorithm. This combined algorithm achieves an average (geometric mean) speedup of 6.4% over the baseline with 2D main memory and the default bus width. The unepic, 482.sphinx, and disparity benchmarks saw the largest speedups (4.5%, 12.9%, and 13.5%, respectively).

Lastly, the fifth bar per benchmark in Figure 7.6 shows the performance of a static bus width of 4,096 bits that was presented in Section 6.2.2. A static bus width of 4,096 bits achieves an average (geometric mean) speedup of 9.7% and is shown to compare my algorithms to the current methodology for using 3D main memory with Wide I/Os where the bus width remains in the highest bus width setting for the entire execution duration. Although the current methodology achieves better performance, as was demonstrated in Section 6.2.3, it comes with an increase an average energy increase of 2.6% (and up to 8.9%).

Based on these performance results, the IPC moving average bandwidth scaling algorithm has the best performance when compared to the oracle algorithm. The IPC moving average

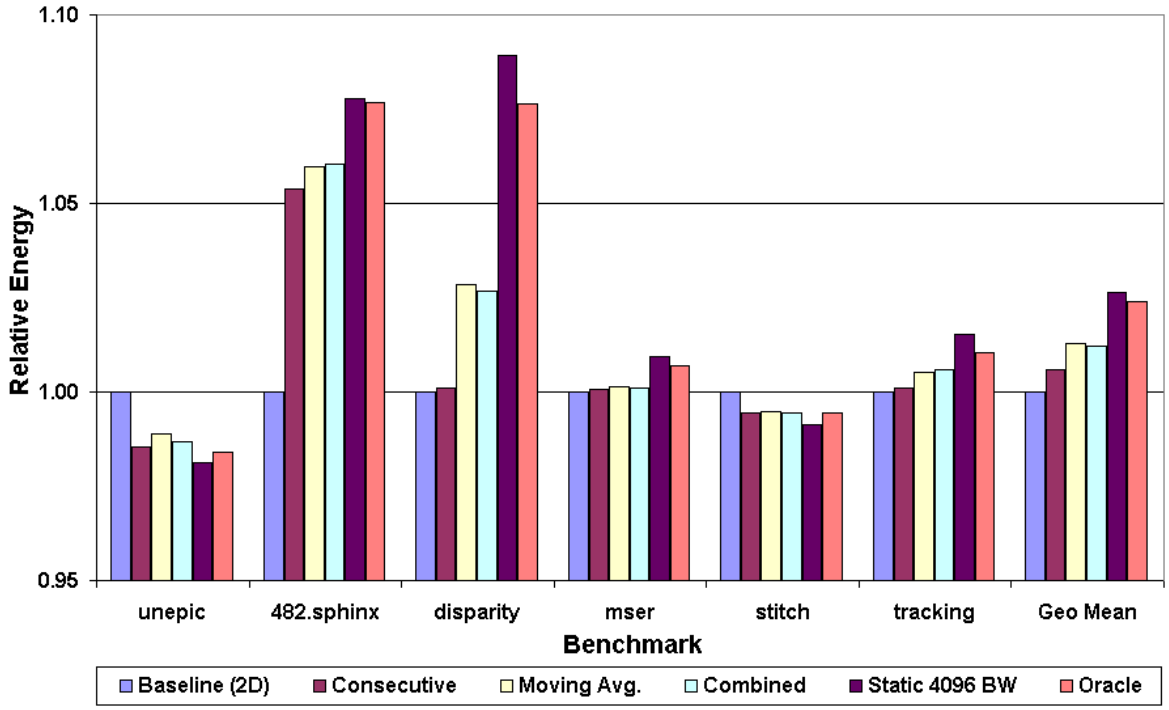

Figure 7.7: DSP energy results of my three bandwidth scaling algorithms on the six benchmarks.

bandwidth scaling algorithm's 6.9% average speedup encompasses the majority of the speedup of both the static bus width setting (9.7%) and the oracle algorithm (10.6%). However, the main motivations for developing my dynamic bandwidth scaling algorithms was not only to increase DSP performance, but also lower the increase in energy consumption by reducing the amount of execution time spent in the maximum main memory bus width setting.

7.3.3 Results – Energy

Figure 7.7 shows the relative energy results of my three dynamic bandwidth scaling algorithms, the static 4,096 bit bus width, and the oracle algorithm with all results normalized to the baseline processor with 2D main memory. The rightmost bar per benchmark shows that the oracle algorithm increases average DSP energy consumption by 2.4%. Although the oracle algorithm

has the best DSP performance (10.6%), it achieves this by having the applications spend an average of 71.8% of their execution time in the maximum bus width setting.

The second bar per benchmark in Figure 7.7 shows the relative energy of the consecutive adjacent L2 line miss bandwidth scaling algorithm. On average, the consecutive adjacent L2 line miss algorithm increases DSP energy consumption by only 0.5% relative to the baseline DSP with traditional 2D DRAM. The consecutive adjacent L2 line miss algorithm achieves this by spending an average of only 25.2% of the execution time in the maximum bus width setting. This leads to better I/O energy efficiency. However, this also explains why it does not achieve the highest speedup of the three bandwidth scaling algorithms (an average performance improvement of 6.6% compared to 6.9% for the IPC moving average algorithm). Lastly, although this algorithm still increases average DSP energy consumption, this is a significant reduction over the static 4,096-bit bus width configuration (fifth bar per benchmark), which increases average DSP energy consumption by 2.6%.

The third bar per benchmark in Figure 7.7 shows the relative energy of the IPC moving average bandwidth scaling algorithm. The IPC moving average algorithm increases DSP energy consumption by an average of 1.3% when compared to the baseline configuration. The IPC moving average algorithm spends an average of 32.1% of the execution time in the maximum bus width configuration, which improves I/O energy efficiency, but still results in a small increase in energy consumption compared to the baseline and consecutive adjacent L2 line miss algorithm, which spends less time in the maximum bus width setting (25.2%). However, the time spent in the higher bus width settings explains why this algorithm achieves the best performance out of the three bandwidth scaling algorithms (an average performance improvement of 6.9%). Although this algorithm also increases average DSP energy consumption, it is still lower than the

static 4,096-bit bus width configuration (fifth bar per benchmark) while retaining most of the performance benefits.

The fourth bar per benchmark in Figure 7.7 shows the relative energy of the combined bandwidth scaling algorithm. The combined algorithm increases energy consumption by an average of 1.2% when compared to the baseline configuration with 2D main memory. The combined algorithm spends 37.1% of its execution time in the maximum bus width configuration, which explains the small increase in DSP energy consumption. However, although it increases energy consumption by a similar amount as the IPC moving average bandwidth scaling algorithm, it achieves a smaller average speedup of 6.4%.

Based on the energy results, we conclude that of my three dynamic bandwidth scaling algorithms, the consecutive adjacent L2 line miss algorithm is the best algorithm for DSP energy consumption. With 3D main memory with Wide I/O and the consecutive adjacent L2 line miss algorithm, DSP energy consumption increases by only 0.5% (compared to 2.6% for the static 4,096-bit bus width) while still increasing average DSP performance by 6.6%. However, if DSP performance is more important, one could conclude that the IPC moving average algorithm is the better dynamic bandwidth scaling algorithm for our processor configuration and benchmarks since it achieves the highest average speedup (6.9%) while increasing DSP energy consumption by 1.3%. If we consider both DSP performance and energy equally in our design and use an energy-delay product, we find the consecutive adjacent L2 line miss algorithm performs better by 0.5%.

Regardless of what algorithm is chosen for our C67x-like DSP utilizing 3D main memory with Wide I/O, it will perform better on the six multimedia benchmarks than the baseline DSP with traditional 2D DRAM while consuming less energy than a configuration where the bus

width statically remains in the widest setting. This is because my algorithms more efficiently scale the effective main memory bus width based on the benchmark's memory needs, which leads to better energy efficiency.

7.4 Summary

In this chapter, I presented a novel contribution, algorithms that can dynamically increase or decrease the main memory bus width of DSPs utilizing 3D main memory with Wide I/O. My dynamic bandwidth scaling algorithms can increase the performance of these DSPs while also addressing the increase in energy that comes from statically remaining in a higher bus width setting for the entire duration of a program, which is what current 3D main memory with Wide I/O does. My algorithms achieve these benefits by dynamically increasing the main memory bus width during memory-intense phases of a program where the wider bus width can be used to fetch adjacent L2 cache lines in advance, but also decreasing bus width during program phases where the wider bus width is not required and would only increase I/O energy. My dynamic bandwidth scaling algorithms can be implemented by adding multiplexers between the interbank datalines and the buffers connected to the I/O pads and the use of memory controller signals to control the multiplexers.

Three dynamic bandwidth scaling algorithms were presented and described including consecutive adjacent L2 line miss, IPC moving average, and an algorithm that combines the previous two approaches. These algorithms use real-time profile data to make their main memory bus width decisions and I found all three could achieve the majority of an oracle scaling algorithm's performance benefit while increasing energy consumption by a small amount compared to the baseline DSP with 2D main memory. The consecutive adjacent L2 line miss algorithm increases average DSP performance by 6.6% (and up to 12.8%) compared to the

baseline DSP with 2D main memory while increasing DSP energy consumption by only 0.5%. The IPC moving average algorithm increases average DSP performance by 6.9% (and up to 14.8%) compared to the baseline DSP with 2D main memory while increasing DSP energy consumption by 1.3%. These results show that my dynamic bandwidth scaling algorithms can be used with 3D main memory with Wide I/O to not only address the growing memory requirements of modern signal processing applications typically run on mobile devices, but can do so without greatly impacting the DSP energy consumption.

As will be described in Chapter 9 (Future Research), I believe more advanced dynamic bandwidth scaling algorithms could be developed using technologies such as neural networks. I also believe the idea could be expanded beyond DSPs and be used to divide the main memory bus among multiple compute resources that are part of the same system (i.e. smart phone) based on the changing memory needs of each compute resource.

The dissertation work described in this chapter was accepted for publication at the IEEE/ACM 2013 International Conference on Computer-Aided Design (ICCAD) [10].

8 Conclusions

3D main memory has been a popular topic of research for the last several years. It was believed that 3D main memory could address the "memory wall" problem by bringing main memory in-package to lower main memory latencies by 45% to 60% and increasing main memory bandwidth with the use of thousands of TSVs. These advantages made 3D main memory an attractive solution for modern embedded DSPs whose multimedia applications are becoming more memory-intensive.

Unfortunately, through my dissertation research it was found that the latency reduction from 3D main memory was much less than previously predicted. The key reason for this was because 2D and 3D main memory systems still had the same DRAM bank organizations and both still retained the same organization of rows and columns. Therefore, the only difference between 2D and 3D main memory was the different interconnections between the DRAM banks and the processor; specifically the latency differences between off-chip metal wires (2D) and TSVs (3D). Through accurate simulations using the Spectre circuit simulator, we found that 3D TSVs reduces main memory access latencies by only 2.4 ns (or 4.1% for our chosen DRAM technology). With these new main memory latencies, I re-evaluated the performance benefits of 3D main memory on a C67x-like DSP and found it to have very little benefit. Based on this finding, I concluded that the key advantage of 3D main memory is the ability to increase main memory bandwidth with the use of additional TSVs. Presenting and re-evaluating this new and more accurate model for 3D main memory latency were the first two contributions of my dissertation research.

The next contribution of my dissertation research was to analyze the current implementation of 3D main memory with Wide I/O, which greatly increases the main memory bus width through

the use of TSVs, but also decrease main memory latencies with two main memory optimizations. Current 3D main memory with Wide I/O has two design modifications that can reduce main memory accesses: 1) removing the DRAM to memory controller synchronizers and 2) placing the address, command, and data pads closer to the DRAM banks. It was found that these two optimizations could decrease main memory latencies by 7.0 ns (or 11.8%) and with the latency savings from going to TSVs, 3D main memory with Wide I/O could reduce main memory latencies by a total of 9.4 ns (or 15.9%). Through system simulation, it was found the current implementation of 3D main memory with Wide I/O improves the performance of a C67x-like DSP by an average of 3.8% (and up to 8.8%) and that further increasing the main memory bus width to 4,096 bits and using adjacent cache-line prefetching increases the performance improvement to an average of 9.7% (and up to 23.3%). Using the increased main memory bus width to fetch additional L2 cache lines greatly benefited the chosen applications because all of the benchmarks were multimedia applications, which have fairly linear main memory access patterns and a high-degree of spatial locality.

Although 3D main memory with Wide I/O improves the performance of embedded DSPs, my dissertation research also found that DSP energy consumption increases by an average of 2.6% (and up to 8.9%). This was because the chosen applications had program phases where the increased bus width and adjacent cache line prefetching were evicting valid L2 cache entries leading to I/O inefficiency. Although 3D main memory with Wide I/O addresses the growing memory requirements of modern signal processing applications, this increase in energy consumption may be unacceptable since embedded DSPs are generally found in mobile devices such as smart phones that have limited battery life and low energy requirements. These results

and observations motivated me towards developing a solution that allowed DSPs to utilize 3D main memory with Wide I/O while keeping energy consumption low.

The final contribution of my dissertation research was proposing algorithms for dynamic bandwidth scaling that dynamically increases the main memory bus width during memory-intense phases in a program, but also decreases the bus width during compute-intense phases to lower I/O energy consumption. Three dynamic bandwidth scaling algorithms were proposed (consecutive adjacent L2 line miss, IPC moving average, and combined) and it was found that the consecutive adjacent L2 line miss algorithm achieves an average DSP performance improvement of 6.6% while increasing energy consumption by only 0.5% and the IPC moving average algorithm achieves an average DSP performance improvement of 6.9% while increasing energy consumption by 1.3%. This is in contrast to the current methodology where the main memory bus width is made wider and kept in this setting, which increases average DSP performance by 9.7%, but increases energy consumption by 2.6%. My dynamic bandwidth scaling algorithm allow DSPs to take advantage of the performance benefits of 3D main memory with Wide I/O, but also reduces the energy impact through better I/O efficiency. It can be implemented by adding multiplexers between the interbank datalines and the buffers connected to the I/O pads and using memory controller signals to control the multiplexers.

As a whole, the models and ideas that I have presented in this dissertation document allow embedded DSPs to take advantage of 3D main memory with Wide I/O to address the memory-intense nature of modern signal processing application, but also meet the stringent energy requirements of mobile computing devices. Previously it was thought the latency reduction and increase in main memory bandwidth were the key advantages of 3D main memory. Although my dissertation research did not disprove the latter, it was shown that increasing the main memory

bus width can also increase DSP energy consumption. Not only do my dynamic bandwidth scaling algorithms allow DSPs to take advantage of the performance benefits of 3D main memory with Wide I/O, but they do so without greatly impacting the DSP energy consumption. These advantages make my dynamic bandwidth scaling algorithms an attractive solution for integrating 3D main memory with Wide I/O and future embedded DSPs found in popular mobile devices.

9 Future Research

My dissertation research proposed several dynamic bandwidth scaling algorithms to improve DSP performance and I/O energy efficiency. However, a number of avenues remain open for future research.

I developed my dynamic bandwidth scaling algorithms through the use of dynamic profile data from the processor taken at intervals of one million instructions. One potential area for future work is in developing more advanced dynamic bandwidth scaling algorithms. For example, additional processor profile data such as ALU usage, load/store queue usage, etc. could be used to design an even more accurate dynamic bandwidth scaling algorithm. Furthermore, an artificial neural network or other form of machine learning could be implemented and after observing the performance changes associated with different main memory bus width settings, in theory, the artificial neural network could learn the behavior of the program and dynamically scale the bus width with even better efficiency than any human designed algorithm using profile data.

Another avenue for future research is implementing a dynamic bandwidth scaling algorithm that uses the program counter (PC) to look for execution patterns. Signal processing applications typically have repetitive execution patterns where the same loops or subroutines are run multiple times across a set of data. For example, image decompression applications typically run the same subroutines across an input image pixel-by-pixel where the pixels are read from the cache using a linear or stride-based access pattern. Therefore, if a dynamic bandwidth scaling algorithm could determine that the application was executing the same subroutines, it could increase the main memory bus width and fetch adjacent L2 cache lines in advance so that the image data would be available in advance leading to an increase in performance.

Lastly, my dissertation research implemented and evaluated dynamic bandwidth scaling algorithms at the single processor-level (DSPs), but many mobile devices are complex systems composed of multiple processors such as general-purpose processors, DSPs, GPUs (graphic processing unit), and ASICs (application-specific integrated circuit). Based on this, another area for future research is investigating the impact of 3D stacked memory and implementing dynamic bandwidth scaling algorithms at the system-level. Since each of the compute resources on a system has vastly different memory requirements, it may be more difficult to develop a dynamic bandwidth scaling algorithm that avoids memory starvation for one or more of the compute resources during the memory-intense phase of one or more of the other compute resources. However, with multiple compute resources to share the main memory bus between, there should be more opportunities for performance and energy benefits making this a promising area for future work.

Bibliography

[1] W. Strauss, "The real DSP chip market," *IEEE Signal Process. Mag.,* vol. 20, pg. 83, 2003.

[2] Electronics Industry Market Research and Knowledge Network, *DSP Market Report,* 2008.

[3] Digital Signal Processors (DSP) Market is Growing at a CAGR of 9.09% & Expected to Reach 9.58 Billion by 2016, 2013. [Online] Available: http://www.prweb.com/releases/digital-signal-processors/market/prweb10563497.htm.

[4] W.A. Wulf and S.A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Computer Architecture News,* vol. 23, pp. 20-24, 1995.

[5] W.R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A.M. Sule, M. Steer, and P.D. Franzon, "Demystifying 3D ICs: The pros and cons of going vertical," *IEEE Design & Test of Computers,* vol. 22, pp. 498-510, 2005.

[6] G.H. Loh, Y. Xie, and B. Black, "Processor design in 3D die-stacking technologies," *IEEE Micro,* vol. 27, pp. 31-48, 2007.

[7] P.J. Denning and S.C. Schwartz, "Properties of the working-set model," *Communications of the ACM*, vol. 15, issue 3, pp. 191-198, 1972.

[8] D.W. Chang, G.-S. Byun, H. Kim, M. Ahn, S. Ryu, N.S. Kim, and M. Schulte, "Reevaluating the Latency Claims of 3D Stacked Memories," *Asia and South Pacific Design Automation Conference,* pp. 657-662, 2013.

[9] D.W. Chang, N.S. Kim, and M. Schulte, "Analyzing the performance and energy impact of 3D memory integration on embedded DSPs," *2011 International Conference on Embedded Computer Systems*, pp. 303-310, 2011.

[10] D.W. Chang, Y.H Son, J.H. Ahn, H. Kim, M. Ahn, M. Schulte, and N.S. Kim, “Dynamic Bandwidth Scaling for Embedded DSPs with 3D-Stacked DRAM and Wide I/O,” *IEEE/ACM 2013 International Conference on Computer-Aided Design,* 2013.

[11] T. Claasen, "An industry perspective on current and future state of the art in system-on-chip (SoC) technology," *Proceedings of the IEEE,* vol. 94, pp. 1121-1137, 2006.

[12] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration," *Proceedings of the IEEE,* vol. 89, pp. 602-633, 2002.

[13] M. Crowley, A. Al-Shamma, D. Bosch, M. Farmwald, L. Fasoli, A. Ilkbahar, M. Johnson, B. Kleveland, T. Lee, and T. Liu, "512 mb PROM with 8 layers of antifuse/diode cells," *IEEE International Solid-State Circuits Conference,* pp. 284-493, 2003.

[14] S.M. Jung, J. Jang, W. Cho, J. Moon, K. Kwak, B. Choi, B. Hwang, H. Lim, J. Jeong, and J. Kim, "The revolutionary and truly 3-dimensional 25F2 SRAM technology with the smallest S3 (stacked single-crystal si) cell, 0.16 um2, and SSTFT (stacked single-crystal thin film transistor) for ultra high density SRAM," *Symposium on VLSI Technology,* pp. 228-229, 2004.

[15] J. Lu, T. Cale, and R. Gutmann, "Wafer-level three-dimensional hyper-integration technology using dielectric adhesive wafer bonding," *Materials for Information Technology,* pp. 405-417, 2005.

[16] BDTI – GPP: Processor Characteristics Relevant to DSP, 2010. [Online] Available: http://www.bdti.com/products/reports_gppproc.htm.

[17] J. Glossner, J. Moreno, M. Moudgill, J. Derby, E. Hokenek, D. Meltzer, U. Shvadron, and M. Ware, "Trends in compilable DSP architecture," *IEEE Workshop on Signal Processing Systems,* pp. 181-199, 2000.

[18] J. Glossner, D. Iancu, M. Moudgill, G. Nacer, S. Jinturkar, S. Stanley, and M. Schulte, "The Sandbridge SB3011 Platform," *EURASIP Journal on Embedded Systems,* vol. 2007, pp. 16, 2007.

[19] J.A. Fisher, P. Faraboschi, and C. Young, Embedded Computing: A VLIW Approach to Architecture, Compilers, and Tools, Morgan Kaufmann, 2005.

[20] R. Banakar, S. Steinke, B.S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad Memory: A Design Alternative for Cache On-chip memory in Embedded Systems," *Proceedings of the 10th International Symposium on Hardware/Software codesign*, pp. 73-78, 2002.

[21] Texas Instruments C67x Specification, 2011. [Online] Available: http://www.ti.com/lit/ds/symlink/tms320c6746.pdf.

[22] B. Jacob, S.W. Ng, and D.T. Wang, "Overview of DRAMs," *Memory Systems: Cache, DRAM, Disk,* Morgan Kaufmann, pp. 316-317, 2007.

[23] Micron. 512 MB: x4, x8, x16 DDR2 SDRAM Features, 2004. [Online] Available: http://download.micron.com/pdf/datasheets/dram/ddr2/512MbDDR2.pdf

[24] R. Anigundi, H. Sun, J.Q. Lu, K. Rose, and T. Zhang, "Architecture design exploration of three-dimensional (3D) integrated DRAM," *Proceedings of the 2009 10th International Symposium on Quality of Electronic Design,* pp. 86-90, 2009.

[25] Y.F. Tsai, Y. Xie, N. Vijaykrishnan, and M.J. Irwin, "Three-dimensional cache design exploration using 3DCacti," *IEEE International Conference on Computer Design,* pp. 519-524, 2005.

[26] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid Cache Architecture with Disparate Memory Technologies," *Proceedings of the 36th International Symposium on Computer Architecture,* pp. 34-45, 2009.

[27] G.H. Loh and M.D. Hill, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture,* pp. 454-464, 2011.

[28] G.H. Loh and M.D. Hill, "Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, pp. 70-78, 2012.

[29] J.S. Kim, C.S. Oh, H. Lee, D. Lee, H.R. Hwang, S. Hwang, B. Na, J. Moon, J.G. Kim, H. Park, J.W. Ryu, K. Park, S.K. Kang, S.Y. Kim, H. Kim, J.M. Bang, H. Cho, M. Jang, C. Han, J.B. Lee, J.S. Choi, and Y.H. Jun, "A 1.2 V 12.8 GB/s 2Gb mobile wide-I/O DRAM with 4× 128 I/Os using TSV-based stacking," *IEEE Journal of Solid-State Circuits,* vol. 47, no. 1, pp. 107-116, 2012.

[30] S. Takaya, M. Nagata, A. Sakai, T. Kariya, S. Uchiyama, H. Kobayashi, and H. Ikeda, "A 100GB/s Wide I/O with 4096b TSVs Through an Active Silicon Interposer with In-Place Waveform Capturing," *2013 IEE International Solid-State Circuits Conference*, pp. 434-435, 2013.

[31] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G.H. Loh, D. McCaule, P. Morrow, D.W. Nelson, and D. Pantuso, "Die stacking (3d) microarchitecture,"

Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 469-479, 2006.

[32] C.C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor-memory performance gap with 3D IC technology," *IEEE Design & Test of Computers,* vol. 22, pp. 556-564, 2005.

[33] G.H. Loh, "3d-stacked memory architectures for multi-core processors," *Proceedings of the 35th International Symposium on Computer Architecture,* pp. 453-464, 2008.

[34] Tezzaron Unveils 3D SRAM, 2005. [Online] Available: http://www.tezzaron.com/about/Press/0501_3d_sram.html.

[35] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinksi, S. Reinhardt, K. Flautner, and T. Mudge, "PicoServer: Using 3D stacking technology to enable a compact energy efficient chip multiprocessor," *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems,* pp. 117-128, 2006.

[36] A. Al Maashri, G. Sun, X. Dong, V. Narayanan, and Y. Xie, "3D GPU architecture using cache stacking: Performance, cost, power and thermal analysis," *Proceedings of the 2009 IEEE International Conference on Computer Design,* pp. 254-259, 2009.

[37] Y. Pan and T. Zhang, "Improving VLIW processor performance using three-dimensional (3D) DRAM stacking," *Proceedings of the 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors,* pp. 38-45, 2009.

[38] Y.F. Sun, C.N. Liu, T.M. Chen, H.C. Hsieh, J.C. Yeh, and Y.C. Chang, "Improvement of Multimedia Performance based on 3-D Stacking Memory Architecture and Software Refinement," *2012 IEEE International Conference on High Performance Computing and Communication*, pp. 1618-1623, 2012.

[39] H.E. Kim, J.S. Yoon, K.D. Hwang, Y.J Kim, J.S. Park, and L.S. Kim, "A Reconfigurable Heterogeneous Multimedia Processor for IC-Stacking on Si-Interposer," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 4, pp. 589-604, 2012.

[40] R. Sampson, M. Ying, S. Wei, C. Chakrabarti, and T.F. Wenisch, "Sonic Milip3De: A Massively Parallel 3D-Stacked Accelerator for 3D Ultrasound," *IEEE 19th International Symposium on High Performance Computer Architecture*, pp. 318-329, 2013.

[41] S. Rixner, W.J. Dally, U.J. Kapasi, P. Mattson, and J.D. Owens, "Memory access scheduling," *Proceedings of the 27th International Symposium on Comptuer Architeccture*, pp. 128-138, 2000.

[42] I. Hur and C. Lin, "Memory Scheduling for Modern Microprocessors," *ACM Transactions on Computer System,* vol. 25 no. 4, pp. 1-36, 2007.

[43] E. Ipek, O. Mutlu, J.F. Martinez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," *Proceedings of the 35th International Symposium on Computer Architecture,* pp. 39-50, 2008.

[44] B. Leibowitz, R. Palmer, J. Poulton, Y. Frans, S. Li, J. Wilson, M. Bucher, A.M. Fuller, J. Eyles, M. Aleksic, T. Greer, and N.M. Nguyen, "A 4.3 GB/s Mobile Memory Interface with Power-Efficient Bandwidth Scaling," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 889-898, 2010.

[45] H. David, C. Fallin, E. Gorbatov, U.R. Hanebutte, and O. Mutl*.,* "Memory Power management via Dynamic Voltage/Frequency Scaling," *Proceedings of the 8th ACM International Conference on Autonomic Computing,* pp. 31- 40, 2011.

[46] D.H. Yoon, M.K. Jeong, and M. Erez, “Adaptive Granularity Memory Systems: A tradeoff between storage efficiency and throughput,” *Proceedings of the 38th International Symposium on Computer Architecture,* pp. 295-306, 2011.

[47] Q. Deng, D. Meisner, L. Ramos, T.F. Wenisch, and R. Bianchini, “MemScale: Active Low-Power modes for Main Memory,” *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 225-238, 2011.

[48] L.N. Chakrapani, J. Gyllenhaal, W. Hwu, S.A. Mahlke, K.V. Palem, and R.M. Rabbah, “Trimaran: An infrastructure for research in instruction-level parallelism,” *Languages and Compilers for High Performance Computing,* pp. 32-41, 2005.

[49] G. Ascia, V. Catania, M. Palesi, and D. Patti, "EPIC-explorer: A parameterized VLIW-based platform framework for design space exploration," *First Workshop on Embedded Systems for Real-Time Multimedia,* pp. 65-72, 2003.

[50] V. Kathail, M. Schlansker, and B.R. Rau, “HPL-PD architecture specification: Version 1.1,” *Technical Report HPL-9380 (R.1),* 2000.

[51] J. Gyllenhaal, W. Hwu, and B.R. Rau, “Hmdes version 2.0 specification,” *Technical Report Impact-96-3*, 1996.

[52] G. Cai and C.H. Lim, “Architectural level power/performance optimization and dynamic power estimation,” *Proceedings of the Cool Chips Tutorial: An Industrial Perspective on Low Power Processor Design in conjunction with MICRO-32*, 1999.

[53] S. Thoziyoor, N. Muralimanohar, J.H. Ahn, and N.P. Jouppi, “CACTI 5.3,” *HP Laboratories Technical Report,* 2008.

[54] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro,* vol. 26, pp. 52-60, 2006.

[55] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Archicture Letters*, pp. 16-19, 2011.

[56] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems," *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture,* pp. 330-335, 1997.

[57] J.L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News,* vol. 34, pp. 17, 2006.

[58] S.K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M.B. Taylor, "SD-VBS: The San Diego Vision Benchmark Suite," *Proceddings of the 2009 IEEE International. Symposium on Workload Characterization*, pp. 55-64, 2009.

[59] Speech at CMU, 2007. [Online] Available: http://www.speech.cs.cmu.edu/.

[60] Apple – iPhone, 2013. [Online] Available: http://www.apple.com/iphone/.

[61] D. Marr and T. Poggio, "Cooperative computation of stereo disparity," *Science*, vol. 194, pp. 283-287, 1976.

[62] P. Viola and M.J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, pp. 137-154, 2004.

[63] R. Szeliski, "Image alignment and stitching: a tutorial," *Foundational Trends in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1-104, 2006.

[64] M.A. Fischler and R.C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381-395, 1981.

[65] J. Shi and C. Tomasi, “Good features to track,” *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593-600, 1994.

[66] B.D. Lucas and T. Kanada, “An Iterative Image Registration Technique with an Application to Stereo Vision,” *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674-679, 1981.

[67] Microsoft Corp. Redmond WA. Kinect for Xbox 360.

[68] G.-S. Byun, Y. Kim, J. Kim, S.-W. Tam, and M..-C. Chang, "An Energy-Efficient and High-Speed Mobile Memory I/O Interface Using Simultaneous Bi-Directional Dual (Base+RF)-Band Signaling," *IEEE Journal of Solid-State Circuits*, vol 47, no. 1, pp. 117-130, 2012.

[69] Tezzaron Semiconductor. Fastack stacking (3D-ICs) technology, 2009.

[70] Ansoft’s 3D full-wave EM filed HFSS Simulator. [Online] Available: http://www.ansys.com/Products/Simulation+Technology/Electromagnetics/High-Performance+Electronic+Design/ANSYS+HFSS.

[71] W.T. Beyene, C. Madden, J.-H. Chun, H. Lee, Y. Frans, B. Leibowitz, K. Chang, N. Kim, T. Wu, G. Yip, and R. Perego, "Advanced modeling and accurate characterization of a 16 Gb/s memory interface," *IEEE Transacations on Advanced Packaging ,* vol. 32, pp. 306-327, 2009.

[72] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," *ACM SIGARCH Computer Architecture News,* pp. 222-233, 1999.

[73] Cadence Virtuoso Spectre Circuit Simulator. [Online] Available: http://www.cadence.com/products/rf/spectre_circuit/pages/default.aspx.

[74] Wide I/O Single Data Rate (Wide I/O SDR). [Online] Available: http://www.jedec.org/standards-documents/docs/jesd229.

[75] The Spice Circuit Simulator. [Online] Available: http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/.

[76] 22nm Low Power PTM Model Site. [Online]. Available: http://ptm.asu.edu/modelcard/LP/22nm_LP.pm.

[77] N.P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers," *Proceedings of the 17th International Symposium on Computer Architecture,* pp. 364-373, 1990.

[78] Synopsys Software and Tools. [Online] Available: http://www.synopsys.com/Tools/Pages/default.aspx.