An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth

Dong Hyuk Woo Nak Hee Seong Dean L. Lewis Hsien-Hsin S. Lee

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
{dhwoo, nhseong, dean, leehs}@ece.gatech.edu

ABSTRACT

Memory bandwidth has become a major performance bottleneck as more and more cores are integrated onto a single die, demanding more and more data from the system memory. Several prior studies have demonstrated that this memory bandwidth problem can be addressed by employing a 3D-stacked memory architecture, which provides a wide, high frequency memory-bus interface. Although previous 3D proposals already provide as much bandwidth as a traditional L2 cache can consume, the dense through-silicon-vias (TSVs) of 3D chip stacks can provide still more bandwidth. In this paper, we contest that we need to re-architect our memory hierarchy, including the L2 cache and DRAM interface, so that it can take full advantage of this massive bandwidth. Our technique, SMART-3D, is a new 3D-stacked memory architecture with a vertical L2 fetch/write-back network using a large array of TSVs. Simply stated, we leverage the TSV bandwidth to hide latency behind very large data transfers. We analyze the design trade-offs for the DRAM arrays, careful enough to avoid compromising the DRAM density because of TSV placement. Moreover, we propose an efficient mechanism to manage the false sharing problem when implementing SMART-3D in a multi-socket system. For single-threaded memory-intensive applications, the SMART-3D architecture achieves speedups from 1.53 to 2.14 over planar designs and from 1.27 to 1.72 over prior 3D designs. We achieve similar speedups for multi-program and multi-threaded workloads on multi-core and multi-socket processors. Furthermore, SMART-3D can even lower the energy consumption in the L2 cache and 3D DRAM for it reduces the total number of row buffer misses.

1. INTRODUCTION

Memory bandwidth, shared by processor cores, GPGPUs, and accelerators, is looming as a major bottleneck for scaling up the performance of modern applications. As more cores are integrated onto a single die [4, 14, 37, 51], the demand for memory bandwidth will grow to unprecedented levels. To alleviate this issue, architects have replaced the traditional, intensely contested and congested front-side bus with new interfaces such as integrated memory controllers, AMD's HyperTransport, and Intel's QuickPath Interconnect. In spite of these innovations, the bandwidth available to future processor will continue to be restricted by the limited number of pins in the processor's package. According to the ITRS, the number of package pins will not grow much in the coming decade—due to cost and power constraints— and most of these additional pins will be delivering power, not data. Therefore, new architectural innovations must be discovered.

One promising solution to this problem is the 3D stacked-DRAM architecture [3, 15, 23, 24, 25], in which the processor cores and main memory are integrated into a single chip stack. Most proposed architectures simply consider the 3D memory to be another level in the cache hierarchy [3], but a recent study demonstrated that a quad-core processor could have as many as sixteen layers of DRAM stacked on top of it without exceeding the maximum thermal limit [24]. In this study, each DRAM layer contained 1GB of memory for a total of 16GB in the stack, more than enough storage to act as the entire main memory for netbook, laptop, and desktop systems. In fact, at 4GB of memory per core, this is even enough memory for most servers; top-of-the-line servers will require advanced liquid-cooling systems to allow for additional DRAM layers. Given this volume of storage capacity, the reduced access latency, and the increased memory bandwidth, stacked DRAM is an excellent candidate for a main system memory [15, 23, 24, 25] in future-generation many-core processors.

Prior studies of 3D stacked memory architectures for increasing memory bandwidth can be grouped into three categories: (1) increasing the memory bus width [15, 23, 24, 25], (2) increasing the frequency of the memory bus and controller [23, 25], and (3) implementing more memory channels [24]. By implementing a memory bus as wide as a cache line that operates at the core's clock frequency, prior proposals have demonstrated that a 3D stacked memory architecture can provide the maximum bandwidth that the L2 cache can consume. In other words, the maximum usable memory bandwidth can be achieved with only a few hundred *through-silicon-vias* (TSVs) between the processor and memory layers (*e.g.*, 512 TSVs for a 64B L2 cache line). This observation suggests that TSV technology is unnecessary because existent system-in-a-package (SiP) processes, which can create a few thousand die-to-die wirebond connections, is sufficient for implementing such a small memory bus.

However, this conclusion relies on one basic assumption: minimizing memory traffic will ensure the best performance. This is a reasonable assumption in a traditional system where main memory is far from the processor, residing somewhere out on the motherboard. However, in a 3D chip stack where memory is on top and quickly accessible, this assumption needs serious reconsideration. We hypothesize that by *increasing* memory traffic, and thus better utilizing the massive bandwidth afforded by high-density TSVs, we can provide useful data in the cache hierarchy more often, significantly improving performance. In other words, instead of designing a traditional *von Neumann bottleneck*-aware processor that minimizes memory traffic, we should design a 3D-aware system that exploits the massive memory bandwidth of a 3D memory-on-chip system.

In this paper, we will demonstrate our hypothesis with a new 3D-

987-1-4244-5659-8/09/$25.00 ©2009 IEEE

aware memory design. In particular, our system leverages the immense bandwidth of a TSV interconnect to eliminate the *trailing-edge effect*, a detrimental side-effect of skinny memory buses that must be occupied for many cycles to transfer a memory block. As a side benefit, our 3D stacked DRAM is also simpler to design than traditional DRAM. Overall, our technique is a very simple design with very high effectiveness, more than halving program execution time on the average.

2. BACKGROUND AND RELATED WORK

2.1 3D IC Stacking Technology

3D integration is a promising new manufacturing technology that allows multiple layers of active silicon to be stacked one on top of the other. The layers are integrated together with TSVs; these vias are short, fast, and dense, allowing for an incredibly high inter-layer bandwidth that simply cannot be matched by other existent technologies like MCMs or SiPs. For example, state-of-the-art TSV manufacturing will be able to produce approximately $4\mu m^2$ vias on a pitch of $4\mu m$ in 2011 [1]. That is several millions of TSVs in one square centimeter. 3D integration also allows for the integration of disparate technologies like CMOS, DRAM, and analog circuits into a single, tightly-coupled chip stack. With such a high concentration of fast connections, the individual layers behave almost as though they were a single layer; that is, inter-layer connections have electrical characteristics similar to standard intra-layer wires, so designers do not have to worry about the excessive loading or transmission-line effects that plague in-package and board-level wiring. At the same time, circuit quality is improved because each layer is manufactured in its own highly-optimized manufacturing process. DRAM arrays are manufactured in a DRAM-optimized process and control logic in a high-speed-CMOS-optimized process; no need to compromise the quality of either circuit type as is presently done for monolithic integration processes like eDRAM [28].

2.2 3D Architectures

Computer architects have been busily exploring new opportunities in the integration of disparate manufacturing technologies [3, 15, 18, 23, 24, 25, 26, 34, 42, 51, 52, 54]. Some of them [15, 23, 24, 25] have focused on how a processor can benefit from enormous memory bandwidth that a 3D-stacked memory can provide, but most of them have not properly addressed how we should re-architect our current memory architecture to take full advantage of the 3D-stacked memory architecture. The only exception is PicoServer [15], which eliminated an L2 cache to improve system throughput. Note that their target application is throughput-sensitive, Tier-1 server applications, which justified their choice of using a 3D DRAM and trading the L2 area for more cores for transactional applications; in contrast, our work focuses on more traditional latency-sensitive, single-threaded applications. Other studies also exploited the benefit of TSVs [16, 19, 27, 53], but the bandwidth was used for an on-chip network among caches or routers.

2.3 The Effect of Limited Memory Bandwidth

To say that the memory bandwidth problem has heavily affected cache design is a severe understatement. Although caches are designed for reducing off-chip bandwidth, Goodman observed that a processor with a cache often consumes higher bandwidth than a processor without one when an application lacks spatial locality [13]. Based on this observation, he proposed a cache design that chiefly exploits temporal locality by using extremely small cache lines. On the other hand, Smith pointed out that a larger cache line is preferable for a system that can provide higher bandwidth [38]. Nonetheless, a larger cache line may aggravate the false sharing problem; several compiler and hardware techniques have been proposed to address this [8, 9, 10, 11, 46]. Toward this issue, Torrellas *et al.* also found that the high miss rate of a large-line cache is due mainly to poor spatial locality, not false sharing [47].

Many prefetching techniques have also been proposed to enhance performance. For this study, we chose to use the Global History Buffer (GHB) stride prefetcher in one of our baselines since the GHB scheme was shown to be the most efficient among many prefetchers [33]. A technique that shares a certain resemblance to the fetching policy used by our 3D memory hierarchy is region prefetching [6, 20, 44, 49, 41]. However, this prefetching method requires complex hardware additions in both the processor and the memory controller (*e.g.*, insertion and scheduling policies) with sophisticated compiler analysis. Furthermore, they still suffer from the trailing-edge effect on the memory bus for fetching large memory blocks. In other words, if a demand fetch request is made during a region prefetch, it must be delayed until the pending region transfer is finished. In this paper, we will show that the high-density TSV bus will fundamentally eliminate the trailing-edge effect once and for all without any of these complicated hardware control mechanisms or software development overheads.

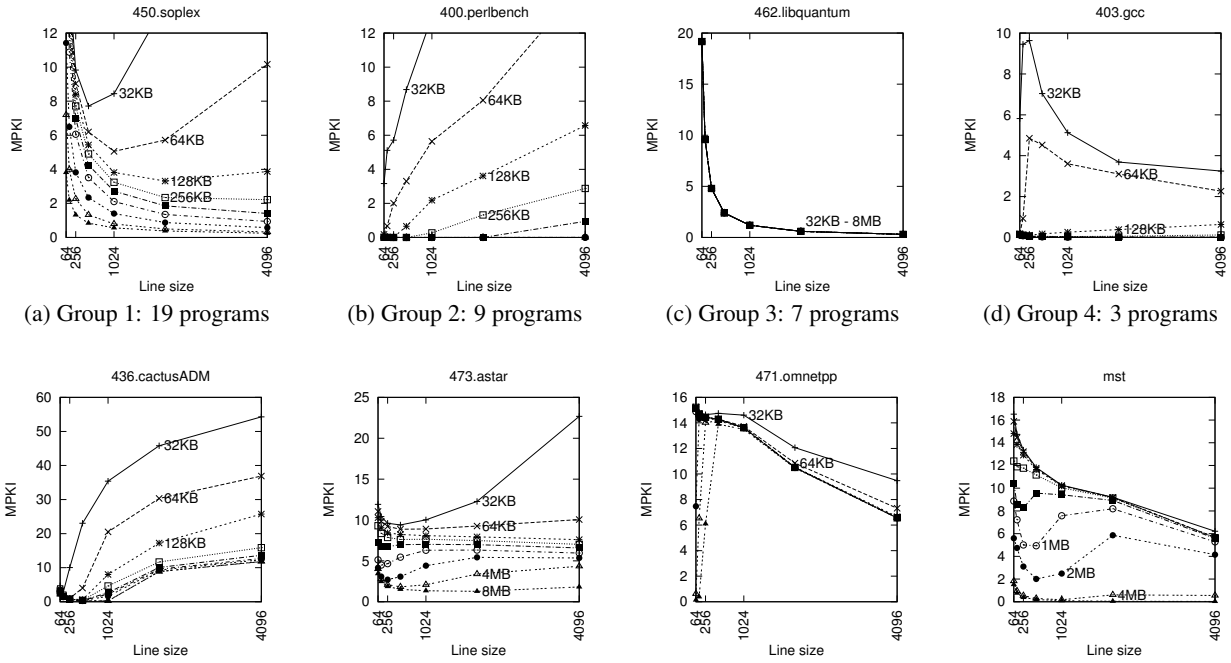
3. 3D DRAM-AWARE PROCESSOR: OPPORTUNITIES AND CHALLENGES

Although 3D memory can provide a huge amount of bandwidth, prior proposals have not fully exploited it for *single-thread applications*. For example, given a four-issue, three-operand 32-bit CISC machine, the worst-case memory requirement will be 48 bytes ($4 \times 3 \times 4$ bytes) per cycle. In this case, any prior 3D DRAM proposals can easily satisfy this bandwidth demand [23, 24, 25]. However, a TSV array can provide as much as 1Tb/sec, given one million TSVs running at 1GHz. Such a bus would be greatly under-utilized when running even the worst single-threaded applications.

There is an old saying: "Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed—you can't bribe God." In this work, we will contend this old saying and quench the latency problem by fully exploiting the potential of TSVs. In particular, we induce many large data transfers to exploit lots of spatial locality. Essentially, we prefetch entire 4KB pages while maintaining a normal 64B line size in the caches for short access latencies and efficient coherence management. It is well known that a large cache line is effective in reducing miss rates [20, 35, 44]. Unfortunately, it also presents several drawbacks. First, the narrow off-chip bus creates the trailing-edge effect, which often degrades the performance of both the requesting processor [5] and the other processors sharing the bus in the system [38]. Second, using a larger line size may pollute the cache due to more conflict misses in a fixed-capacity cache [38]. Finally, larger line sizes may worsen the false sharing problem. As we will demonstrate, we can use 3D integration and a revised DRAM design to eliminate the trailing-edge penalty of large memory fetches. At the same time, the impact of conflict misses is reduced by the shorter latency of our technique. We also show that performance degradation due to false sharing can be minimized in a multi-socket system.

3.1 Revisiting Line Size Effect Without Off-Chip Pin Count Constraint

First we revisit the effect of cache line size on modern application performance in order to better understand why a re-architecting for 3D stacked memory is needed. We consider a cache line size



(a) Group 1: 19 programs (b) Group 2: 9 programs (c) Group 3: 7 programs (d) Group 4: 3 programs

(e) Group 5: 1 program (f) Group 6: 1 program (g) Group 7: 1 program (h) Group 8: 1 program

Figure 1: MPKIs of Cache Memories of Different Capacities and Line Sizes

ranging from 64B to 4KB (the size of a page, the maximum line size without software modification), much larger than those in prior studies [13, 38], and a cache capacity from 32KB to 8MB. For applications, we use memory traces taken from forty-two applications from the SPEC2006int, SPEC2006fp, Olden, and NU-MineBench benchmark suites. Figure 1 plots the misses per thousand instructions (MPKI) as a function of line size. Each curve represents a particular cache capacity. For brevity, we categorized applications with similar MPKI behavior into eight groups and show only one application per group.

Figure 1(a) shows the MPKI of 450.soplex, which represents nineteen applications (Group 1). When the cache is small (*e.g.*, 32KB), this group's MPKI initially decreases but then quickly increases as the line size grows. These results confirm the common wisdom of the cache pollution effect. Importantly, though, we find this wisdom does not hold for very large caches. Instead, the number of misses continues to decrease with increasing line size for a cache exceeding a certain capacity (256KB for 450.soplex). Therefore, once the cache memory becomes sufficiently large, capacity misses are no longer a problem and spatial locality rules the day. Even more interestingly, the MPKI of a small cache with a large line size is much lower than that of a large cache with a small line size. For example, in Figure 1(a), the MPKI of a 256KB cache with a 4KB line is lower than that of an 8MB cache with a 64B line.

In stark contrast to Group 1, the MPKI of Group 2 in Figure 1(b) monotonically increases with line sizes. However, MPKI is only sensitive to the line size of a small cache; beyond a certain capacity (512KB for 400.perlbench) the MPKI metric levels off. This phenomenon likely occurs because the working set necessarily fits into a sufficiently large cache. In contrast, the MPKI of Group 3 decreases with line size in Figure 1(c). Clearly, they do not have good temporal locality but instead have very good spatial locality; thus, a larger cache line is always beneficial. Group 4 in Figure 1(d) shows that the line size matters only for small caches. Finally, the remaining four applications have unique characteristics. The MPKI of 436.cactusADM (Figure 1(e)) increases as the line size increases. Similarly, the MPKI of 473.astar (Figure 1(f)) increases as the line size increases when the cache is large (1MB to 4MB). These are mainly due to very poor spatial locality. In contrast, 471.omnetpp (Figure 1(g)) and mst (Figure 1(h)) both interestingly show that a 4KB line size works well enough for sufficiently large caches (*e.g.*, 1MB or 2MB) when compared to the 64B baseline.

In summary, most modern applications will benefit from a smaller L1 cache line size,[1] but a larger cache line is found to be helpful for a much larger L2 cache. Interestingly, the maximum line size, an entire page (4KB), is found to be very effective in a large cache. In general, these results, based on today's complex applications, are consistent with observations made on very old, simple programs [38]. Unfortunately, such a large line is not practical in a traditional processor because the trailing-edge effect negates any gain from the reduced miss rate (demonstrated in Section 4). However, it is very attractive in a 3D stacked memory architecture because the extremely wide TSV bus can eliminate all trailing-edge effect and its associated penalty.

3.2 Leveraging the Additional Dimension to Implement a Wide Bus

Although enlarging the line size to 4KB helps reduce the miss rate, it may degrade the overall performance if it significantly increases the access latency. To estimate this, we used Cacti 5 [45] to estimate the access latency of an eight-way 1MB cache for different line sizes. Unfortunately, we found that access latency increases almost linearly with the line size. Therefore, it will not be wise to expect an overall performance improvement by simply enlarging the line size.

To explain the latency increase, we review the common cache subarray design employed by Cacti [50]. Large cache arrays are often subdivided into smaller subarrays to minimize latency and power. Figure 2 illustrates the layout of a cache bank that has been

[1] Throughout this paper, we use 64B line size for the L1 cache.

partitioned into multiple subbanks (four in this case). Each subbank is also partitioned into multiple mats (four in the figure), wherein all arrays in a single mat share predecoding logic. Each mat is connected to the cache interface using an H-tree distribution network, also shown in Figure 2. For cache read, write, fill, and write-back operations, addresses and data are routed through the H-tree between the cache controller interface and the target subbank (the figure illustrates a line written into four mats of a subbank from the controller using white arrows). The complete cache access latency depends both on the number of subarrays in a row and in a column. For example, an eight-way 1MB cache with large 4KB lines consists of thirty-two rows and 32,768 columns. Such a highly skewed layout fails to balance the length of horizontal and vertical wires in the H-tree. Furthermore, a 4KB-line cache requires an extremely wide H-tree network resulting in a long signaling delay. Clearly, simply enlarging the cache line size up to 4KB is not a good design choice.

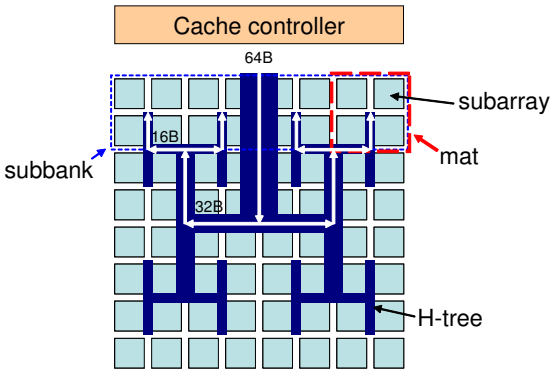


Figure 2: Cache Subarray Design

So instead of enlarging the line size, we chose to implement a fetch size which is different from the line size, (*e.g.*, fetching a 4KB page upon a miss into sixty-four 64B cache lines). Note that using different line and fetch sizes has been investigated, such as next N-line prefetching [39, 40]. However, Przybylski found that the optimal fetch size depends on the latency and the width of a memory channel [35]. Furthermore, he found that the overhead of fetching a memory block larger than a line, incurring a trailing-edge effect penalty, renders such aggressive fetching useless in the memory systems of the early 90s. But a couple years later, Temam and Jegou were able to exploit larger recent cache sizes by proposing the Virtual Line Scheme to fetch a page into a secondary cache for improving numerical applications [44]. Despite of the trailing-edge effect, the large secondary cache alleviated the cache pollution problem. Of course, the increased memory traffic of these schemes has been a big concern for these fetch policies, so software-directed schemes were also proposed to work around the bus contention issue. More recently, integrated memory controllers ave made region prefetching feasible [20, 49], in which a processor (with help from the compiler) monitors the status of a memory channel and prefetches a page only when the channel is idle. This necessarily complicates both hardware and compiler design. It is now time, given the unprecedented bandwidth of emerging 3D-stacking technology, to revisit these aggressive fetching scheme.

The first fetching policy we evaluate is fetching a 4KB page in one transfer, buffering it, and then filling each 64B chunk through a 64B-wide H-tree network in the L2 cache. This policy aligns with our observation that a smaller line is desirable for L1 and a larger one for L2. As shown in Section 3.1, fetching 4KB lines to large L2 caches substantially reduces L2 MPKI, so optimizing the H-tree for L1 miss latency (*i.e.*, 64B-wide H-tree) makes much more

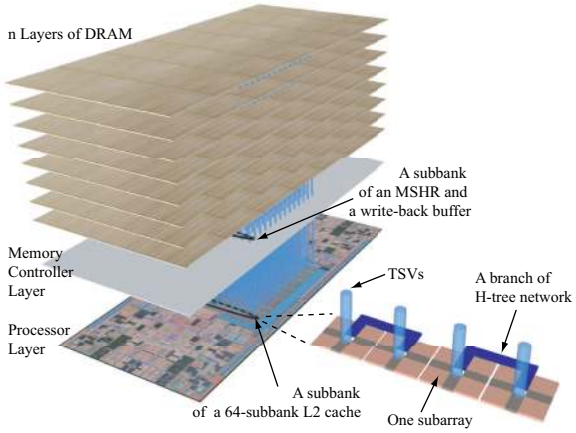


Figure 3: SMART-3D Memory Hierarchy: Implementing a 64B-Wide Bus with TSVs directly on top of Each L2 Subbank

sense than optimizing for the latency of comparatively infrequent L2 misses.

Unfortunately, filling 4KB over a 64B H-tree incurs a trailing-edge penalty in the L2 cache itself (instead of the memory bus) because we can write only one 64B chunk of data each cycle. In other words, filling 4KB of data will consume sixty-four cycles of the L2 cache bandwidth, preventing the processor from accessing the L2 cache for a significant period of time. Note that, such intra-cache trailing-edge effects are also a problem for prior schemes [20, 44, 49]. To overcome this, we propose *SMART-3D*, a Stacked Memory-Aware, Rich TSV-enabled 3D memory hierarchy, which uses a conventional, planar 64B H-tree for read and write operations from the L1 and a vertical 4KB bus for cache fill and write-back operations to the 3D main memory. To realize such design, we place a second CMOS layer between the processor and the DRAM stack (Figure 3); this layer contains a memory controller, an L2 miss status handling register (MSHR), and an L2 write-back buffer. In order to provide 4KB of data without trailing-edge penalties, we partition the L2 cache, L2 MSHR, and write-back buffer into sixty-four subbanks each, allowing for sixty-four simultaneous 64B operations between the L2 and 3D memory. These sixty-four buses are implemented directly on top of each subbank with TSVs, connecting each L2 subbank with its respective subbank of the MSHR and that of the write-back buffer. In this cache design, a read or a write operation uses the conventional H-tree network, while a fill or a write-back operation uses the new TSV bus as shown in the figure. Filling a 4KB page into sixty-four subbanks in this way requires interleaving of cache sets, which in turn necessitates some modification to the predecoding logic.

However, from the Cacti analysis, our sixty-four subbank design slightly underperforms the optimal number of subbanks for the four L2 sizes we simulated. As shown in Table 1, the access latency of our 64-subbank cache will be about one or two cycles slower (given a 3GHz clock) than an optimal cache.

Table 1: The Access Latency of the 8-Way Cache (Cacti 5)

Capacity	1MB	2MB	4MB	8MB
Optimal latency (ns) (The optimal # of subbanks)	1.75 (4)	1.95 (4)	2.65 (8)	4.56 (16)
The latency of 64-subbank cache (ns)	2.20	2.53	3.17	5.01

This new SMART-3D memory design has several interesting implications. The first is related to the data eviction policy. As we

fetch sixty-four cache lines simultaneously, we can choose the eviction victim at a 64B or 4KB granularity. We name these two policies the *local LRU policy* and the *global LRU policy*. We simulated both policies and found that the results are application-specific. Significantly though, we found that the MPKI difference between these two is negligible. The biggest MPKI difference is approximately 0.33. Because the local LRU policy can generate write-back traffic for sixty-four different lines, each potentially to a different page, we opt for the global LRU policy, simplifying the design of our MSHR and write-back buffer. In particular, upon an L2 miss, the L2 cache controller needs to check the centralized control array of the MSHR and the write-back buffer instead of checking sixty-four different control bit arrays. Effectively, we use a conventional associative lookup process but distributed data arrays. Furthermore, choosing the global LRU policy allows us to reduce the size of the tag array, similar to Sector Cache [22, 31].

The next issue is invalidation traffic in an inclusive L2 cache. If the inclusion property is maintained between the L1 and L2, an eviction from the L2 will invalidate the same line, if exists, in the L1. In our SMART-3D design, one L2 eviction may yield up to sixty-four invalidations in the L1. Fortunately, as we will show in Section 4, the effect of such invalidations is insignificant. In this paper, we assume that each L2 line maintains an inclusion bit [48] to minimize the amount of unnecessary eviction traffic sent to the L1.

The third issue is the tight coupling between each L2 bank and its associated memory controller. Such coupling necessitates an architecture in which all cores on the chip share multiple L2 cache banks through some interconnect network, as shown in Figure 4. Such an architecture is widely used in commercial multi-core products [17], GPGPUs [21] and a recently proposed 3D memory system [24].

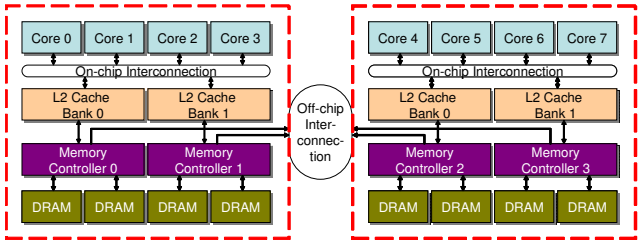


Figure 4: Overall System Architecture

3.3 Novel DRAM Design Issues

In addition to cache design issues, we also carefully evaluated the design of the DRAM layers because our SMART-3D requires 32kbits of TSVs and the area of each TSV is non-trivial (2μm $\times$ 2μm with 4μm pitch according to ITRS projection for 2011 [1]). Furthermore, we do not want to compromise the density of each DRAM array accommodating the TSVs.

First of all, we assume a 256Mbit DRAM array tile (32nm) that consists of 2^{14} rows where each row has 2^{14} bits. We also assume that the height of each DRAM tile is extended by 40% for sense amplifiers while the width of it is extended by 20% for a row decoder. The first design shown in Figure 5(a) is based on a standard DRAM design where each tile has a row decoder and a column decoder. For this design, we carefully matched the TSV pitches with the width of DRAM tile and found that only 256 TSVs are allowed within the given width of a DRAM tile. In other words, to support 32k TSVs, we need 16×8 tiles (Figure 5(a)), which amounts to $29.36mm \times 12.58mm$ in area (including TSV area overhead). In this design, we use the white space on our memory controller layer to route from the distributed memory TSVs into the center of a chip to connect to the TSVs going to the cache on the processor die as shown in Figure 3. In this design, multiple ranks can be implemented vertically by sharing the same TSV bus across the several DRAM layers. However, we found that, in this DRAM design, an L2 miss will look up 128 DRAM tiles simultaneously, potentially consuming a considerable amount of power. Note that the current off-chip DRAM design also necessitates such multiple array lookup due to a large off-chip pin pitch compared to the width of each DRAM chip. For example, to form a 64b data bus, we need eight DRAM chips in a DRAM module where each DRAM chip outputs eight bits. In this configuration, eight different DRAM chips form one DRAM rank. Thus, upon a request, eight DRAM chips are accessed in parallel.

We evaluated a second design option shown in Figure 5(b). In this design, we place all TSVs in the middle of the DRAM die. In other words, 128 tiles share a common TSV bus in the center. The width of eight DRAM tiles will accommodate 2048 TSVs, thus 16 rows are needed to place all 32K TSVs. Using the same method of estimation, this design consumes approximately $29.36mm \times 12.77mm$. Similarly, we can build multiple ranks by vertically stacking these DRAM layers, all sharing the same TSV bus. Because the row buffer in each tile is 2KB (2^{14} bit), only two tiles are selected upon an L2 miss, and no column address (*i.e.*, CAS select) and column mux will be required. This design requires global routing in the DRAM layer, but we only enable two DRAM tiles upon an L2 miss, saving a significant amount of power over the first design.

The third option is a variation of the second design as shown in Figure 5(c). In this design, we split one DRAM layer of the second design into four layers. Each of them uses only their portion of 8kbits of TSVs for data transfer, but all of them still need space for accommodating 32kbits of TSVs. Clearly, this design requires a larger overall space (four layers combined) but reduces the footprint of individual die ($14.68mm \times 6.68mm$). Furthermore, wire length between each tile and the TSVs decreases compared to the second design.

3.4 Support for Multi-Socket Platforms

One very interesting problem created by SMART-3D is potential exacerbation of the false sharing problem. Note, this is not a problem for a single-die multi-core SMART-3D processor as it still has an L1 with 64B lines and the L2 is shared. Yet false sharing could become worse in a multi-socket SMART-3D system. As Figure 4 shows, each quad-core socket has its own memory controller and 3D DRAM. In essence, the aggregated DRAM from these two sockets form a non-uniform memory access (NUMA) machine. Data are freely shared and moved across DRAM located in different sockets. For a SMART-3D processor, fetching a page from remote memory will suffer from the trailing-edge effect because of the requisitely narrow inter-socket communications bus. Also, the probability of having falsely-shared data will be significantly higher given the L2 line size, potentially creating a performance-crushing ping-pong effect between the two processors.

To address this issue, we propose an adaptive SMART-3D design[2] that suppresses the page-fetching policy (1) if the target page is mapped to the memory space of a remote socket or (2) if any line of the target page is cached in the L2 on a remote socket. De-

[2]The line management in SMART-3D is similar to subblocking techniques [2, 8, 10], but, instead of invalidating subblocks within a line, our adaptive fetching scheme uses the same principle to suppress prefetching multiple 64B L2 lines from main memory. Our technique can be used with the subblocking techniques although it is not evaluated in this paper.

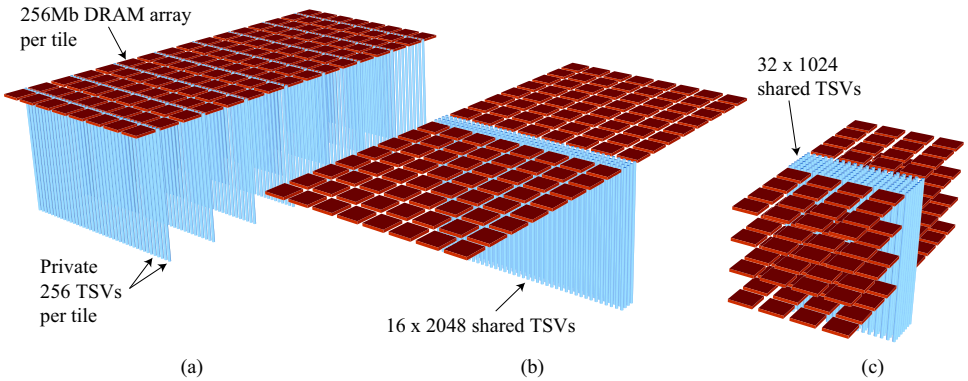


Figure 5: Different DRAM Designs. (a) Private TSVs per Tile, (b) Shared TSVs, and (c) Folded DRAM with Shared TSVs

Table 2: Baseline Processor Configurations

Clock frequency	3.0 GHz	
Processor model	14-stage, OoO, 4-wide fetch/issue/retire	
ROB size	192	
Physical register file size	128 (INT) / 128 (FP)	
Branch predictor	Hybrid (16K global / local / meta tables), 2K BTB, 32-entry RAS	
L1 I cache	2-port 2-way, 64B-line, 32KB, LRU, 1-cycle, 8-entry MSHR	
L1 D cache	2-port 4-way, 64B-line, 32KB write-back, LRU, 2-cycle latency, 1-cycle throughput, 8-entry MSHR	
Unified L2 cache	2-port 8-way, 64B-line, 1MB inclusive write-back, LRU, 6-cycle latency, 1-cycle throughput, 8-entry MSHR	
Memory	2D Base	350-cycle minimum latency, 8B-wide bus, 800MHz double data rate bus, 12.8GB/sec
	3D Base	250-cycle minimum latency, 64B-wide bus, 3GHz, 192GB/sec

tecting the first condition is easy because an L2 cache just needs to test whether the requested physical address is in its local memory space. On the other hand, to detect the second condition, we adopt a page-level counting Bloom filter similar to RegionScout [29] or Page Sharing Table [12]. Upon allocating or deallocating a 64B line, the page-level counting Bloom filter is updated to keep its counter values consistent with the corresponding cache tag array.

Now we describe the miss handling process of our proposed scheme. Note that the line size we employ is 64B, not 4KB, so all coherence messages are handled at that granularity. Furthermore, without loss of generality, we assume the MESI protocol. Upon an L2 miss, the L2 finds a victim page using the global LRU policy, allocates space (sixty-four 64B lines), and creates MSHR entries for any line not already cached. The miss generates a typical MESI protocol message, triggering lookups in other L2 caches (we use an inclusive cache for simplicity). When a processor receives this coherence request, it looks up both its L2 and its page-level counting Bloom filter, responding with the conventional MESI message as well as the 1-bit datum from the filter.

Once all responses reach the requesting processor, this processor will execute the appropriate memory fetch as detailed next. Case 1: the required page is mapped to the local memory and no part of the page is being shared. The entire page is fetched into the requestor's L2 (just as in the single-processor case). The requested line is marked Exclusive (read) or Modified (write) while the other sixty-three lines are marked Exclusive. Case 2: the requested line is mapped to local memory and not shared, but the Bloom filter bit from at least one remote processor indicates that at least one line from the page is cached. The requestor fetches only the requested line from local memory and places it in its L2. Case 3: the requested line is mapped to remote memory. This line alone is placed in the local L2. Case 4: another processor delivers the requested line. This line alone is placed in the local L2.

Should a new demand miss be generated to the same page while a previous miss is still being processed, we have designed our adaptive SMART-3D to perform as follows: (1) If the previous miss is waiting for coherence responses from other processors, this new demand miss generates a conventional coherence message. (2) If the previous miss is already fetching the entire page from the main memory, then this new miss request just waits for this page to be fetched. (3) If the previous miss is fetching a 64B line alone, then this new miss request initiates a second, separate miss handling process.

Finally, the requesting processor may itself receive a request message from a second requester that may happen to request (1) a line from a page that is waiting for the Bloom filter data of other processors, or (2) a line from a page that is already being fetched from main memory. In (1), the initial requester gives up attempting to fetch an entire page and instead settles for a single line. In (2), the initial requester responds to the second requester after it has completed caching the target line.

Clearly, an adaptive SMART-3D cache will often fail to fetch an entire page, leaving invalid lines. These lines effectively reduce the capacity of the L2 cache, a downside of our adaptive scheme. In our design, if a page is partially allocated and the L2 cache encounters a miss to an invalid line in this partial page, the L2 cache initiates the conventional miss handling process for the missing 64B line and fills the line either from other processors or from the main memory.

4. EVALUATION

4.1 Simulation Framework

We evaluate our SMART-3D using SESC [36]. To evaluate single-threaded applications on a single-core processor and multiple applications on a multi-core system, we used the SPEC2006, NU-MineBench, and Olden benchmark suites. To evaluate multi-threaded applications on both single- and a dual-socket multi-core systems, we use the SPLASH-2 benchmark. We have excluded applications that fail to cross-compile or use unsupported system calls.

We compare our SMART-3D architecture against several other

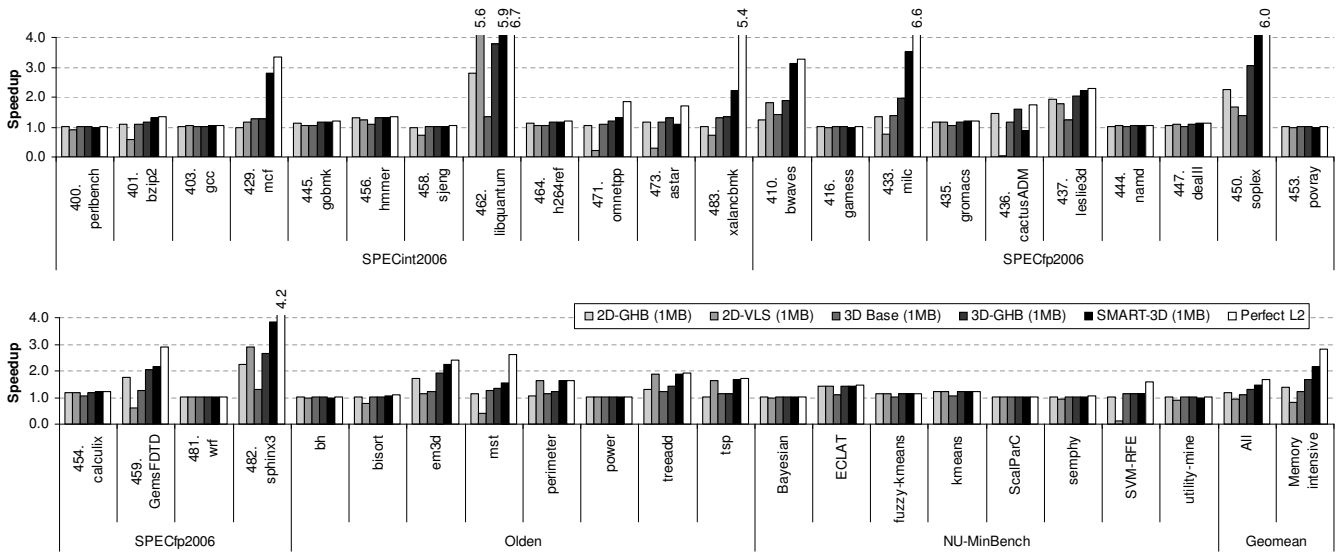


Figure 6: Performance Improvement of Single-Threaded Applications

machines. The parameters for the 2D and 3D baseline processors are listed in Table 2. We enumerate all the machine models in the following bullets. Based on these machine models, we also varied the sizes of cache capacity as detailed subsequently. Their access latencies were estimated using Cacti 5 [45] and are listed in Table 3.

- **2D Base**: A barebones 2D processor with off-chip DRAM modules. Its system parameters are listed in Table 2.
- **2D-GHB**: A *2D Base* with a PC/CS *Global History Buffer* (GHB) prefetcher [30]. The size of the GHB prefetch buffer is 32KB.
- **2D-VLS**: A *2D Base* processor with *Virtual Line Scheme* that employs and fetches 4KB cache lines [44].
- **3D Base**: A barebones 3D processor with DRAM layers stacked directly atop [23, 25]. The parameters are listed in Table 2. This model also assumes a reduced memory latency of a true-3D memory [24].
- **3D-GHB**: A *3D Base* with a PC/CS GHB prefetcher [30]. The size of the GHB prefetch buffer is 32KB.
- **SMART-3D**: Our Stacked Memory-Aware, Rich TSV-enabled 3D processor.

Table 3: Access Latency of an 8-Way Cache (in cycles of a 3GHz clock, U: unused in this paper)

	64 KB	128 KB	256 KB	512 KB	1 MB	2 MB	4 MB	8 MB	16 MB
Optimal $	U	U	U	U	6	6	9	15	22
64-subbank $	5	6	6	6	7	8	10	16	U

Note that we aggressively assume the memory bus of 3D Base runs at core clock frequency with cache-line size width and is fully pipelined [23, 25]. Thus, 3D Base can return one cache line back to the L2 on per-core-clock cycle basis. As we will show later, such an aggressive design makes the utilization of the bus very low. Given this observation, we did not further evaluate an even more aggressive mechanism using multiple channels, which also burdens the area of having multiple memory controllers. However, note that our proposal can support multiple channels as shown in Figure 4, thus orthogonal to such multiple channel implementations. In addition to such aggressive bandwidth model, we assumed a 30% reduction in memory latency for 3D Base to imitate the benefit gained by using a true-3D memory model [24].

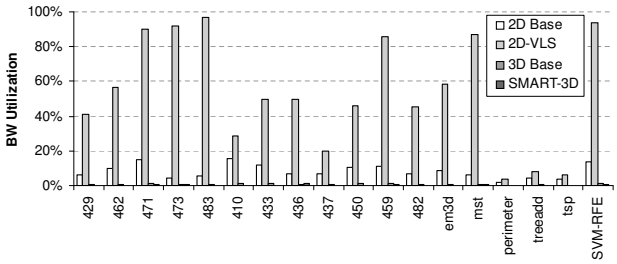


Figure 7: Bandwidth Utilization of Memory-Intensive Apps

4.2 Single-Core Performance

4.2.1 Overall Performance Evaluation

First, we report the speedup of various machine models in Figure 6 for all single-thread benchmark programs. Each bar shown was normalized to the 2D Base model (=1.0). The *Memory-intensive* category is the collection of those applications with more than 1.5x speedup on the *Perfect L2* model. As shown, the benefit of simple DRAM stacking (3D Base) is rather limited for single-thread applications. The average speedup of *3D Base* is only 1.25x and 1.12x for Memory-intensive applications and All, respectively. Further, it only achieved 44.1% (Memory-intensive) and 67.4% (All) of the performance of the ideal Perfect L2 model. On the other hand, the average speedup of our SMART-3D is 2.14x (Memory-intensive) and 1.46x (All), capturing 75.9% (Memory-intensive) and 88.1% (All) of the Perfect L2 performance. As shown, 3D Base can be improved with a prefetcher (bars shown in 3D-GHB); however, even with this area investment, the speedup is only about half what SMART-3D achieves. Fair-capacity results will be discussed later. Another noteworthy point is that, as explained previously, the access latency of the SMART-3D 64-subbank cache is one cycle longer (seven cycles total) than that of 3D Base. In spite of this penalty, it still achieves much higher performance than 3D Base. Also shown are the results of a 2D-VLS machine (Virtual Line Scheme [44])

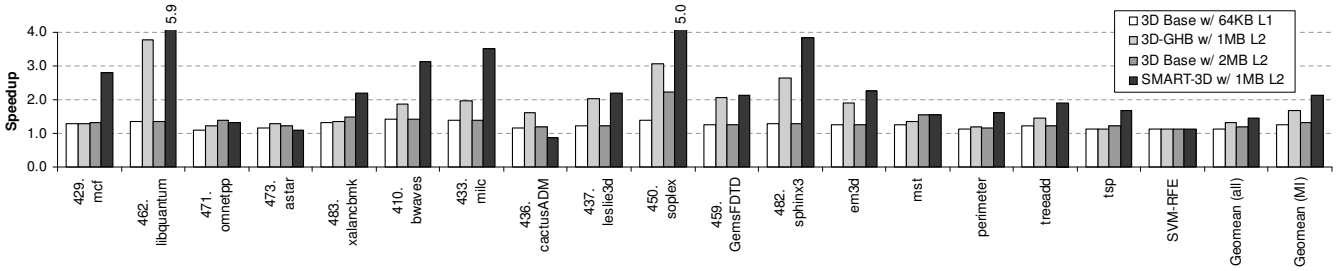


Figure 8: Performance and Area Comparison

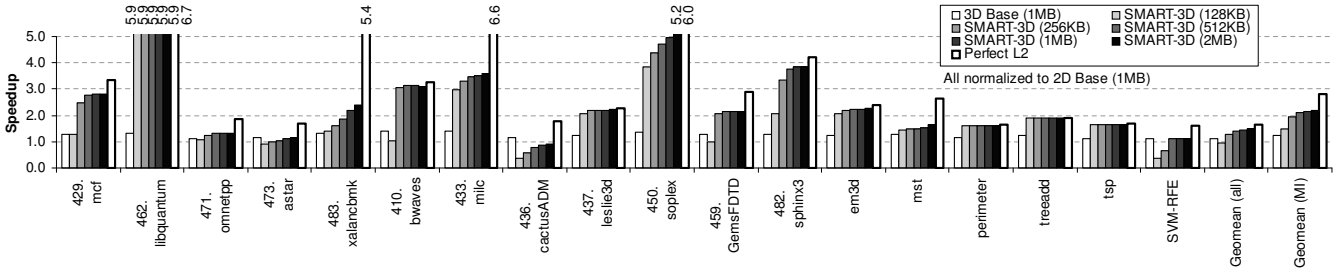


Figure 9: Performance Evaluation of Cache Size Sensitivity for SMART-3D

that fetches a page upon an L2 miss. The original VLS work assumed one-cycle and two-cycle latencies for the L1 and L2. The latter is no longer appropriate for the sheer size of modern L2 and was updated to six cycles. As shown in Figure 6, 2D-VLS, in fact, degrades performance by 8.1% on average due to the trailing-edge penalties. These results demonstrate that a conventional architecture with restricted memory bandwidth cannot utilize a 4KB line size even when the workloads have good spatial locality. To further understand this issue, Figure 7 plots bandwidth utilization for memory-intensive applications. In fact, 2D-VLS nearly saturates its maximum bandwidth in the majority of applications. The figure also shows that the bandwidth provided by 3D Base is more than enough for single-thread applications (no application is bandwidth-bound). Finally, SMART 3D also shows low bandwidth utilization in the figure although it generates more data traffic on the memory bus. This is simply because the overall bandwidth provided by the TSV bus is massive, compared to the amount of data fetched. This also suggests that there is plenty of room on the TSV bus to fulfill larger data demand when running multiple applications on a multi-core system.

One drawback of SMART-3D is the increase in conflict misses as mentioned earlier. However, we found that 473.astar and 436.cactusADM are the only two memory-intensive applications with an increased MPKI (by 17.7% and 158.9%). That is why their SMART-3D speed-ups are lower than those for 3D Base in Figure 6. Note that these applications could be optimized for SMART-3D by regrouping the data to enhance page-level locality.

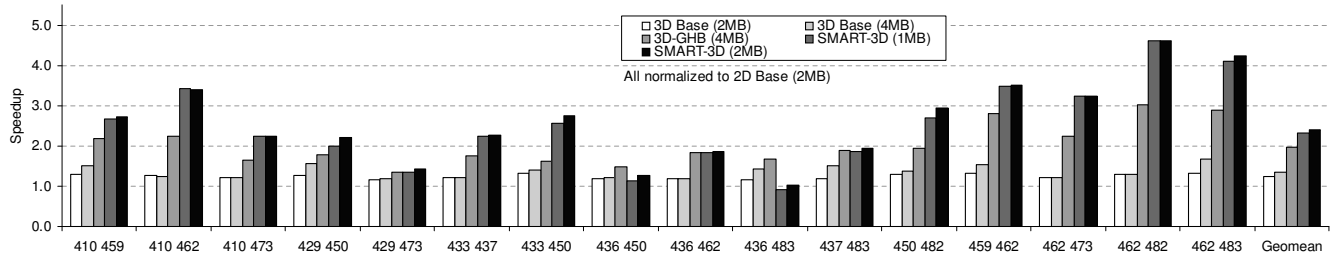
Lastly, we must consider that SMART-3D may generate more L1 invalidations as an inclusive-cache machine. In our experiments, we found the number of L1 invalidations is very small compared to the number of L1 accesses (2.6% in the worst case, 471.omnetpp). That is, when a line is evicted from the L2, there is a good chance that all the corresponding L1 lines have already been evicted from the L1 cache. Further, we also examined the increase in the L1 miss rate due to such invalidations. In most applications, the L1 miss rate is barely affected. 436.cactusADM suffers the most, but still the L1 miss rate increases by only 1.4%. On average, the L1 miss rate is increased by just 0.04%, skewed, obviously, by 436.cactusADM.

4.2.2 Performance and Area Comparison

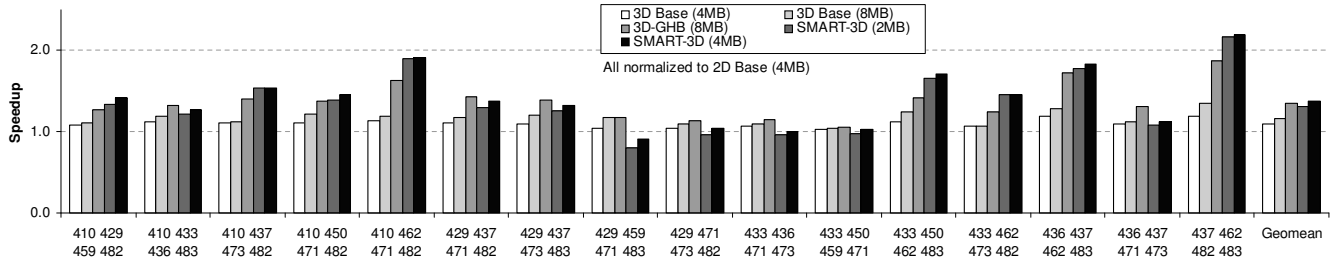
So far, we have assumed a non-blocking L2 with an 8-entry MSHR, but for SMART-3D, we require more area to keep eight 64B lines in each bank of our 64-subbank MSHR. The area of our 64-bank MSHR, approximately 32KB, is sixty-four times larger than that in the baseline. To perform fair comparison in terms of area cost, we considered the following processor models in Figure 8: (1) 3D Base with 64KB data L1, (2) 3D Base with 32KB GHB prefetch buffer, (3) 3D Base with an enlarged 2MB L2 (conservative comparison), and (4) SMART-3D. Note that the areas are not exactly the same but close. All base cases have a 1MB L2. We normalize the performance to that of 2D Base. For the GHB machine, a GHB prefetcher maintains 256-entry global history and fetches four lines (a prefetch depth of four) to its dedicated 32KB prefetch buffer. For brevity, we show results for memory-intensive applications only but report the averages for all applications in Figure 8. On average, expanding the L1 data cache to 64KB (3D Base w/ 64KB L1) and using a GHB (3D-GHB w/ 1MB L2) improves the performance by 12.1% and 31.2%, repsectively. More interestingly, the performance improvement for 3D Base w/ 2MB L2 falls in between at 18.0% while SMART-3D w/ 1MB L2 is 46.1%. These results show that just enlarging the L2 is not worth the area cost; a considerable portion of the extra capacity stores data is not reused.

4.2.3 Evaluation of Cache Size Sensitivity

In this section, we experiment a variety of L2 sizes for SMART-3D. A larger cache will reduce conflict misses but requires a longer access latency as shown in Table 3. Figure 9 shows the speedup numbers for SMART-3D on an eight-way L2 from 128KB to 2MB. We also show 3D Base as a reference point. As with Figure 8, results are normalized to 2D Base. As shown, SMART-3D with a 256KB L2 outperforms 3D Base with a, much larger, 1MB L2 in most cases (excepting 473.astar, 436.cactusADM, and SVM-RFE). In the case of 462.libquantum and 450.soplex, SMART-3D with 256KB L2 achieves speedups of 5.90x and 4.38x, respectively. Moreover, SMART-3D with a 256KB L2 performs almost as well as Perfect L2 for 437.leslie3d, perimeter, treeadd, and tsp. When raising the cache capacity to 512KB, 1MB, and 2MB,



(a) 2-Core System (Baseline: 2D Base with 2MB)



(b) 4-Core System (Baseline: 2D Base with 4MB)

Figure 10: Performance Improvement of Multi-Program Workloads on a Multi-Core Processor

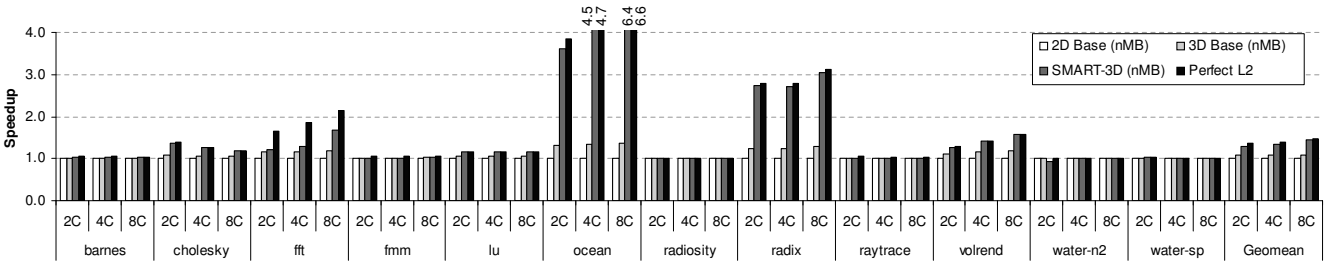


Figure 11: Performance Improvement of Multi-Threaded Workloads on a Multi-Core Processor

the SMART-3D will achieve 73.9%, 75.8%, and 77.4% of the performance of Perfect L2.

4.3 Multi-Core Performance

Next, we evaluate the performance of a multi-core processor that is running several single-threaded applications simultaneously. Here, we assume similar the same processor models as in Table 2 except that the L2 cache is shared among cores as shown in Figure 4. Since simulating all permutations of all forty-two applications in the suites is impractical, we formed sixteen groups. Each group contains randomly selected memory-intensive applications from SPEC2006. Figure 10 shows the simulation results of these groups on a two-core and four-core system. In these experiments, the L2 size varies depending on the number of cores. The baseline (=1.0) is a 2D Base machine with nMB L2 where n is the number of cores. We also conducted experiments for the following configurations: (1) 3D Base with an nMB cache, (2) 3D Base with a $2n$MB cache, (3) 3D Base with a $2n$MB cache and a GHB prefetcher, (4) SMART-3D with a $0.5n$MB cache, and (5) SMART-3D with an nMB cache.

As shown in Figure 10(a), on a two-core system, the SMART-3D with a 2MB L2 outperforms all other models except 436.cactusADM / 450.soplex and 436.cactusADM / 483.xalancbmk. As explained earlier, 436.cactusADM has very bad spatial locality, so its performance is degraded with our scheme. However, when it is scheduled with 462.libquantum, the overall performance is still improved mainly due to the improvement of 462.libquantum. In many cases, even a 1MB L2 SMART-3D can outperform a 4MB 3D-GHB. On average, SMART-3D 1MB and 2MB achieves 2.31x and 2.40x speedup for a two-core whereas the best performing alternative, a 4MB 3D-GHB, only reaches 1.97x.

Similar trend is found in Figure 10(b) on four-core systems—a smaller SMART-3D 4MB outperforms the larger 3D-GHB 8MB in many cases. One outlier foursome was found: the performance of 429.mcf / 459.GemsFDTD / 471.omnetpp / 483.xalancbmk on SMART-3D is lower than that on 2D Base even though each shows a decreased L2 MPKI when run individually on a single-core system. Here, the cache contention among these applications resulted in an overall performance degradation, suggesting a new cache partitioning strategy may be needed; this is beyond the scope of this paper. On average, a SMART-3D 4MB machine improves performance by 37.1% for a quad-core while a 3D-GHB 8MB improves it by 35.6%. We also evaluated an eight-core system, but do not show the results for lack of space. A SMART-3D 8MB which have 1.44x over the 1.42x achieved by a 3D-GHB 16MB.

Next we consider multi-threaded applications from the SPLASH-2 benchmark suite. Here again we scale the L2 capacity with the number of cores. Figure 11 shows the simulation results on a two-

Table 4: Relative Traffic of SMART-3D (Normalized to 3D Baseline)

Benchmark	429	462	471	473	483	410	433	436	437	450	459	482	em3d	mst	peri-meter	tree-add	tsp	SVM-RFE
L2 fill	4.77	1.01	27.98	75.60	22.38	1.00	5.91	158.98	1.62	3.05	17.04	2.19	3.64	38.51	1.03	1.04	1.06	59.38
L2 write-back	8.10	1.00	6.00	55.51	38.35	1.00	4.69	182.38	1.83	2.05	3.48	2.33	53.28	1.95	1.00	1.01	1.05	52.27
Row miss	0.20	0.10	0.98	1.51	1.11	0.15	0.31	3.56	0.09	0.10	0.49	0.28	1.63	0.90	0.31	0.02	0.05	0.98

, four-, and eight-core processor. Our SMART-3D outperforms both 2D Base and 3D Base, especially for memory-intensive applications. Although our proposed scheme often degrades performance when running fmm and water-n2, which are computation-intensive, our design generally improves the performance of multi-threaded applications.

4.4 Dual-Socket Performance

To evaluate our *Adaptive SMART-3D design* as proposed in Section 3.4, we performed simulations for a dual-socket system. Because our simulation infrastructure does not support a multi-level coherence protocol, we simply model a two-socket single-core system connected through an off-chip bus. We assume write-through L1 caches and use the MESI protocol between the L2 caches on different sockets. For the perfect L2, we assumed that every single access to the perfect cache is a cache hit regardless of whether the line is shared with or modified by other processor. We use binary files compiled by a conventional compiler that is not optimized for NUMA architectures, and our OS allocates a newly-requested page to the memory space of the requesting processor.

As shown in Figure 12, our Adaptive SMART-3D improves the performance of memory-intensive applications well. For example, our design achieves 3.63x speedup for ocean. Not surprisingly, it is also found that our scheme may degrade performance for computation-intensive applications, 20% degradation in the worst case (raytrace). We carefully reviewed the results and found that the L2 miss rate had increased from 0.4% to 0.8% on one socket and from 0.32% to 1.06% on the other socket (one reason why the L2 miss rate is so low is that our L1 is write-through, so a lot of write-through traffic accounts for L2 hits.) Since raytrace is compute-bound, even a small increment in L2 miss rate turns out to be a significant impact to the overall performance. Furthermore, we profiled the memory trace for raytrace and found that 59% of pages are shared between these two processors. However, out of sixty-four cache lines in each page, only 7.56 cache lines are actually shared on average. This indicates a need for a SMART-3D-aware compiler to aggregate the shared lines to minimize unnecessary misses.

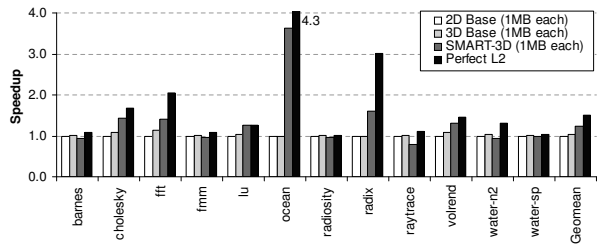


Figure 12: Performance Improvement of Multi-Threaded Workloads on Two Processors in a Different Socket

4.5 Energy Consumption Analysis

The power consumed by switching a TSV is several orders of magnitude lower than an off-chip I/O pin [32]. In other words, because of power supply and packaging constraints, it is simply impossible, in a conventional 2D system, to bring in an entire page at once (to eliminate the trailing-edge effect) by adding enough pins. We simply cannot afford the required power before we put in a desirable number of pins on package.

With respect to the 3D baseline design, we found an interesting trade-off between the 3D baseline and our SMART-3D. Table 4 shows the relative traffic of SMART-3D normalized to 3D Base when running a memory-intensive application on a single-core processor with a 1MB L2 cache and 64-bank 4GB DRAM. Clearly, our new fetching scheme may bring in unwanted data. However, we also found that the total number of row buffer misses (*i.e.*, DRAM array look-ups) of our SMART-3D is much lower than that of the 3D baseline in the majority of the cases, leading to potential energy advantage using SMART-3D. The low row buffer hit rate of the 3D baseline can simply be attributed to its multiple access streams going to different rows of a memory bank. Even if these accesses have good spatial locality, two (or more) memory streams can be issued to different rows alternately in the same memory bank, causing row buffer misses all the time. On the other hand, in SMART-3D, each L2 miss and its induced write-back collapse these otherwise finer-grained accesses (64B each) into one larger transaction (4KB), thereby reducing the total number of row buffer misses.

Our simulation results show that the average row buffer hit rate of memory-intensive applications in *3D Base* is 55.05%. This suggests that we only use two 64B lines before closing a DRAM row even though other cache lines in this row buffer may be accessed in a near future. In contrast, SMART-3D fetches an entire row into the L2 cache, eliminating potentially redundant DRAM array lookups (note that aggregate row buffer capacity is usually smaller than the capacity of L2 cache).

To evaluate the energy implication, we modeled energy consumption of an L2 cache using Cacti 5 [45] (Table 5). For the energy model of the SMART-3D cache, we modeled our cache as writing and reading all sixty-four subbanks for fill and write-back operations, respectively. We also conservatively modeled the energy consumed by the TSVs for fill and write-back operations by extracting TSV capacitance using Synopsys Raphael [43]. Interestingly, in a conventional subarray-based cache design, most of the energy is consumed in the H-tree network, not in the subbanks. Consequently, the energy consumed by a fill or a write-back operation in our 64-subbank cache is not high.

Table 5: Energy Values from Cacti 5 (nJ) (including TSV Energy for SMART-3D cache)

	Read	Write	Fill	Write-back
Baseline (8-way 1MB) $	0.28	0.27	0.27	0.28
SMART-3D (64-subbank 8-way 1MB) $	0.30	0.32	0.53	0.83

Additionally, we used Cacti 5 to model the energy consumption of commodity DRAM arrays. We modeled each tile as 32nm 256Mb array as explained in Section 3.3 and found that each tile consumes 34.85 nJ upon a row buffer miss. The DRAM architecture of 3D Base assumes each tile has their own private 256-bit TSV. Hence, it requires two DRAM tile accesses to obtain a 64B

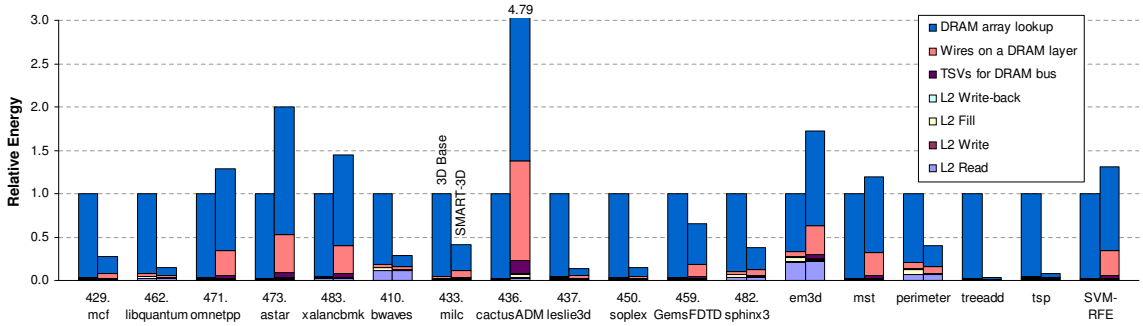


Figure 13: Dynamic Energy Consumption of L2 Cache, DRAM Bus, and DRAM Arrays

line. We also modeled the energy consumption of the wires connecting each DRAM tile to the TSVs in the middle of each die using the global interconnect model of the Predictive Technology Model [7]. Here, we assumed that each DRAM layer is 1GB as shown in Figure 5(c). Based on these assumptions, we estimated that these wires consume 0.32 nJ and 20.78 nJ upon a DRAM access for 3D Base and SMART-3D, respectively. Furthermore, we modeled the energy consumption of the TSVs between the memory controller layer and four DRAM layers (4GB). Upon a DRAM access, we estimated that 3D-Base and SMART-3D consume 0.02 nJ and 1.34 nJ in these TSVs, respectively.

Figure 13 shows the dynamic energy breakdown with respect to the L2 cache, DRAM bus, and DRAM arrays when running memory-intensive applications on a processor with a 1MB L2 cache and 64-bank 4GB DRAM. As shown in the figure, a large amount of energy is consumed by the DRAM arrays when we run memory-intensive applications. As explained previously, because SMART-3D reduces the number of row buffer misses, it saves a lot of energy in many applications. However, when we run an application that does not have good spatial locality, *e.g.*, 473.astar and 436.cactusADM, SMART-3D consumes a large amount of energy in DRAM bus (due to more L2 misses and larger data transfer upon an L2 miss) and in DRAM arrays (due to more row buffer misses). On average (geomean), we found that these three components of SMART-3D consume 47% of energy of 3D-base. We also performed another estimation with a shared TSV design for 3D Base, *i.e.*, fetching 64B from one DRAM tile, and found that these three components of SMART-3D consume 80% of energy of 3D Base.

5. CONCLUSION

In this paper, we demonstrated that simply stacking main memory on top of a processor die does not exploit the full potential of 3D-IC technology; in particular, it falls short on improving the performance of single-thread applications. Toward this goal, we re-architected the L2 cache and its interface to the 3D stacked DRAM, proposing SMART-3D to ameliorate latency by exploiting the excessive, high-density bandwidth of TSV between the processor last-level cache and the 3D DRAM. Upon each L2 miss, the SMART-3D architecture fetches an entire page of data but keeps the caches organized by 64B lines to avoid complicating coherency. We also proposed an adaptive SMART-3D design to mitigate the false sharing problem in a multi-socket system. Finally, we analyzed the design trade-off for the DRAM interface without compromising the DRAM density with the TSV placement. We evaluated SMART-3D with single-threaded, multi-programmed, multi-threaded, and multi-socket workloads, and we found that our design improves performance significantly. On average, for single-threaded memory-intensive applications, the speedups range from 1.53 to 2.14 compared to a conventional 2D architecture and from 1.27 to 1.72 compared to prior 3D-stacked memory designs. Furthermore, as our analysis showed, SMART-3D can even lower the energy consumption in the L2 cache and 3D DRAM for it reduces the total number of row buffer misses.

6. ACKNOWLEDGMENT

This research is supported in part by an NSF grant CCF-0811738 and the Center for Circuit and System Solutions (C2S2), a research center under Focus Center Research Program sponsored by Semiconductor Research Corporation. The authors would also like to thank anonymous reviewers and Dae Hyun Kim at Georgia Tech for their comments and assistance on the final version of the paper.

7. REFERENCES

[1] International Technology Roadmap for Semiconductors, 2007.

[2] C. Anderson and J. Baer. Two Techniques for Improving Performance on Bus-Based Multiprocessors. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 1995.

[3] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die Stacking (3D) Microarchitecture. In *Proceedings of the International Symposium on Microarchitecture*, 2006.

[4] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th Design Automation Conference*, 2007.

[5] D. Burger, J. Goodman, and A. Kägi. Memory Bandwidth Limitations of Future Microprocessors. In *Proceedings of the International Symposium on Computer Architecture*, 1996.

[6] J. Cantin, M. Lipasti, and J. Smith. Stealth Prefetching. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[7] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu. New paradigm of predictive MOSFET and interconnect modeling for early circuit simulation. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 201–204, 2000.

[8] Y. Chen and M. Dubois. Cache Protocols with Partial Block Invalidations. In *Proceedings of Seventh International Parallel Processing Symposium*, 1993.

[9] C. Dubnicki and T. LeBlanc. Adjustable Block Size Coherent Caches. In *Proceedings of the International Symposium on Computer Architecture*, 1992.

[10] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenström. The Detection and Elimination of Useless Misses in Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, 1995.

[11] S. Dwarkadas, P. Keleher, A. Cox, and W. Zwaenepoel. Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology. In *Proceedings of the International Symposium on Computer Architecture*, 1993.

[12] M. Ekman, P. Stenström, and F. Dahlgren. TLB and Snoop Energy-Reduction using Virtual Caches in Low-Power Chip-Multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2002.

[13] J. R. Goodman. Using cache memory to reduce processor-memory traffic. In *Proceedings of the International Symposium on Computer Architecture*, 1983.

[14] J. H. Kelm, D. R. Johnson, M. R. Johnson, B. Tuohy, N. Crago, A. Mahesri, S. S. Lumetta, M. I. Frank, and S. J. Patel. Rigel: An Architecture and Scalable Programming Interface for a 1000-core Accelerator. In *Proceedings of the International Symposium on Computer Architecture*, 2009.

[15] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. PicoServer: Using 3D Stacking Technology To Enable A Compact Energy Efficient Chip Multiprocessor. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[16] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. Yousif, and C. Das. A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures. In *Proceedings of the International Symposium on Computer Architecture*, 2007.

[17] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-Way Multithreaded SPARC Processor. *IEEE micro*, 25(2):21–29, 2005.

[18] D. L. Lewis and H.-H. S. Lee. Architectural Evaluation of 3D Stacked RRAM Caches. In *IEEE International 3D System Integration Conference*, 2009.

[19] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *Proceedings of the International Symposium on Computer Architecture*, 2006.

[20] W. Lin, S. Reinhardt, and D. Burger. Reducing DRAM Latencies with an Integrated Memory Hierarchy Design. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2001.

[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro*, 28(2):39–55, 2008.

[22] J. S. Liptay. Structural Aspects of the System/360 Model 85, Part II: The Cache. *IBM Systems Journal*, 7(1):15–21, 1968.

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

[24] G. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *Proceedings of the International Symposium on Computer Architecture*, 2008.

[25] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee. A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy. In *Proceedings of the 43rd Annual Conference on Design Automation*, 2006.

[26] N. Madan and R. Balasubramonian. Leveraging 3D Technology for Improved Reliability. In *Proceedings of the International Symposium on Microarchitecture*, 2007.

[27] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell. Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2009.

[28] R. E. Matick and S. E. Schuster. Logic-based eDRAM: Origins and rationale for use. *IBM Journal of Research and Development*, 49(1):145, 2005.

[29] A. Moshovos. RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence. In *Proceedings of the International Symposium on Computer Architecture*, volume 32, page 234. IEEE Computer Society; 1999, 2005.

[30] K. Nesbit and J. Smith. Data Cache Prefetching Using a Global History Buffer. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2004.

[31] H. Olnowich. Set associative sector cache, Jan. 8 1985. US Patent 4,493,026.

[32] R. Patti. 3D-ICs: The Evolution and Direction of a New Technology, *IEEE International 3D Systems Integration Conference*. 2009.

[33] D. G. Perez, G. Mouchard, and O. Temam. MicroLib: A Case for the Quantitative Comparison of Micro-Architecture Mechanisms. In *Proceedings of the International Symposium on Microarchitecture*, 2004.

[34] M. Petracca, B. G. Lee, K. Bergman, and L. P. Carloni. Photonic NOCs: System-Level Design Exploration. *IEEE MICRO*, 29(4), 2009.

[35] S. Przybylski. The Performance Impact of Block Sizes and Fetch Strategies. In *Proceedings of the International Symposium on Computer Architecture*, 1990.

[36] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[37] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a Many-Core x86 Architecture for Visual Computing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 2008.

[38] A. Smith. Line (block) Size Choice for CPU Cache Memories. *IEEE Transactions on Computers*, 36(9):1063–1076, 1987.

[39] A. J. Smith. Sequential Program Prefetching in Memory Hierarchies. *IEEE Computer*, 1978.

[40] A. J. Smith. Cache Memories. *ACM Comput. Surv.*, 14(3):473–530, 1982.

[41] S. Somogyi, T. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial Memory Streaming. In *Proceedings of the International Symposium on Computer Architecture*, 2006.

[42] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2009.

[43] Synopsis. "Raphael". http://www.synopsys.com.

[44] O. Temam and Y. Jegou. Using Virtual Lines to Enhance Locality Exploitation. In *Proceedings of the International Conference on Supercomputing*, 1994.

[45] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. Jouppi. CACTI 5.1. *HP Laboratories, Palo Alto, Technical Report*, 20, 2008.

[46] J. Torrellas, M. Lam, and J. Hennessy. Shared Data Placement Optimizations to Reduce Multiprocessor Cache Miss Rates. In *Parallel Processing: International Conference: Selected Papers.*, 1990.

[47] J. Torrellas, M. Lam, and J. Hennessy. False Sharing and Spatial Locality in Multiprocessor Caches. *IEEE Transactions on Computers*, 43(6):651–663, 1994.

[48] W.-H. Wang, J.-L. Baer, and H. M. Levy. Organization and Performance of a Two-Level Virtual-Real Cache Hierarchy. In *Proceedings of the International Symposium on Computer Architecture*, 1989.

[49] Z. Wang, D. Burger, S. Reinhardt, K. McKinley, and C. Weems. Guided Region Prefetching: A Cooperative Hardware/Software Approach. In *Proceedings of the International Symposium on Computer Architecture*, volume 30, 2003.

[50] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. *Digital WRL Research Report 93/5*, 1994.

[51] D. H. Woo, J. B. Fryman, A. D. Knies, M. Eng, and H.-H. S. Lee. POD: A 3D-Integrated Broad-Purpose Acceleration Layer. *IEEE Micro*, July/August, 2008.

[52] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid Cache Architecture with Disparate Memory Technologies. In *Proceedings of the International Symposium on Computer Architecture*, 2009.

[53] Y. Xu, Y. Du, B. Zhao, X. Zhou, Y. Zhang, and J. Yang. A Low-Radix and Low-Diameter 3D Interconnection Network Design. In *Proceedings of the International Symposium on High Performance Computer Architecture*, 2009.

[54] W. Zhang and T. Li. Microarchitecture soft error vulnerability characterization and mitigation under 3d integration technology. In *Proceedings of the International Symposium on Microarchitecture*, 2008.