PicoServer Revisited: On the Profitability of Eliminating Intermediate Cache Levels

Prateek Tandon[†] prateekt@umich.edu
Jichuan Chang[‡] jichuan.chang@hp.com
Ronald G. Dreslinski[†] rdreslin@umich.edu
Parthasarathy Ranganathan[‡] partha.ranganathan@hp.com
Trevor Mudge[†] tnm@umich.edu
Thomas F. Wenisch[†] twenisch@umich.edu

[†]*Advanced Computer Architecture Lab, University of Michigan* [‡]*HP Labs*

Abstract

The confluence of 3D stacking, emerging dense memory technologies, and low-voltage throughput-oriented many-core processors has sparked interest in single-chip servers as building blocks for scalable data-centric system design. These chips encapsulate an entire memory hierarchy within a 3D-stacked multi-die package. Stacking alters key assumptions of conventional hierarchy design, drastically increasing cross-layer bandwidth and reducing the latency ratio between successive layers. Hence, prior work, specifically PicoServer, suggests flattening the hierarchy, eliding intermediate caches that otherwise lengthen the critical path between L1 and stacked memory.

Although PicoServer argues for a flattened memory hierarchy for web serving workloads, it remains unclear if its conclusions hold more generally—particularly when considering metrics besides access latency—and more recent studies have often included intermediate caches. In this paper, we investigate the bandwidth*,* latency*, and* energy filtering *afforded by an L2 cache. For data-centric scientific and server applications, we conclude: (1) 3D stacking provides copious bandwidth, hence L2 bandwidth filtering is moot; (2) although a flat hierarchy is optimal for access patterns with poor temporal locality, some workloads benefit from access latency reduction afforded by L2, an effect magnified by latency-intolerant in-order cores and for memory technologies with asymmetric read and write latencies; and (3) intermediate caches are rarely desirable from an energy perspective, and only if the cache is optimized for low leakage.*

1. Introduction

The confluence of 3D stacking of logic and memory, emerging dense memory technologies, and low-voltage throughput-oriented many-core processors has sparked interest in single-chip servers as building blocks for scalable data-centric system design [23]. These single-chip servers encapsulate an entire memory hierarchy within a 3D-stacked multi-die package. However, 3D stacking alters key assumptions of conventional memory hierarchy design. For example, cross-layer bandwidth is drastically increased, and the latency ratio between successive layers is reduced [16]. In light of these inflections, prior work has suggested flattening the memory hierarchy, eliding intermediate caches that otherwise lengthen the critical path between L1 and stacked memory. For example, *PicoServer* proposes removing the L2 and reallocating the area to additional cores on the chip [10].

Although the *PicoServer* study demonstrates both performance and energy advantages of a flattened memory hierarchy for web serving workloads, many recent studies of 3D stacked systems nevertheless have included intermediate cache levels between L1 and a stacked memory [5, 20, 27] or stacked last-level-cache [16, 17]. The lack of clarity regarding the utility of intermediate caches stems partially from uncertainty about the ultimate performance and energy characteristics of 3D stacked devices. For example, Loh describes how the physical design of DRAM can substantially alter its performance [16]. Liu and co-authors describe alternative 3D stacked DRAM-based design configurations that have up to a 2x difference in performance [15]. Similar uncertainty exists with other potential memory technologies; a survey by Lee and co-authors indicates that published phase-change memory (PCM) performance estimates vary by up to 1.45x [13]. As another example of this uncertainty, the latency values cited by Li and co-authors [14] and Venkataraman and co-authors [26] vary by up to 2.5x for PCM.

In this paper, we revisit the conclusions of the *PicoServer* study to identify the inflection points where an intermediate cache level becomes profitable. Whereas *PicoServer* focuses primarily on the latency impact of intermediate caches versus a flat hierarchy, it has also been observed that caches can act as bandwidth filters [7], and, more recently, as energy filters [12, 25] for lower hierarchy levels. Hence, we consider the utility of an intermediate cache from all three perspectives, *bandwidth*, *latency*, and *energy filtering*. In addition, we investigate design tradeoffs when memory read and write latencies are asymmetric, as is common for emerging memory technologies. We find memory parameter inflection points for a suite of eight applications, four from the SPEC

CPU2006 suite (two each with good and poor relative temporal locality), and four from emerging data-centric workloads that motivate single-chip server design [23].

We make the following contributions:

- We characterize the bandwidth requirements of our workload suite, both with and without an L2 cache, and demonstrate that bandwidth filtering is moot given the large bandwidth provided by 3D-stacked memories. For these benchmarks, the highest bandwidth requirement without an L2 is approximately 29 GB/s with eight high-end cores, a requirement easily fulfilled by two DDR3 channels, and at least 5x below the projected bandwidth capability of 3D stacked memory.
- We show that although a flat hierarchy is optimal for access patterns with poor temporal locality, some scientific and data-centric workloads benefit from access latency reduction provided by intermediate cache levels, an effect that is magnified by latency-intolerant in-order cores. We show that half the data-centric workloads do not require an L2 unless the stacked memory access latency is well over 12x the L2 access latency.
- We find that an L2 cache is rarely desirable from an energy perspective for data-centric workloads, even when the ratio of main memory to L2 dynamic access energy is high. Importantly, we find that it is critical for the L2 to have low leakage power for it to provide any energy gain, as L2 leakage can rapidly offset any energy savings achieved from conserving dynamic power.

The remainder of this paper is organized as follows: Section 2 discusses related work, and Section 3 describes our experimental methodology. We consider the bandwidth, latency, and energy filtering impact of intermediate caches in Section 4, Section 5, and Section 6 respectively. In Section 7, we conclude.

2. Related Work

Our study is inspired by the disparity between the recommendation of *PicoServer* [10] to flatten the memory hierarchy and other more recent studies of 3D stacked memory hierarchies [5, 16, 17, 20, 27] which retain intermediate caches. *PicoServer* proposes an architecture where multiple simple, throughput-optimized, in-order processing cores are 3D-stacked with multiple DRAM dies that comprise main memory. *PicoServer* flattens the memory hierarchy by removing the L2 and re-allocating the area for additional cores. For the web-tier applications studied, the L2 provides little benefit, and the throughput advantage of re-allocating the area for additional cores is greater.

The *PicoServer* study focuses primarily on the latency impact of an L2 cache, observing that it slows the critical access path to the main memory. Bandwidth and energy filtering have been suggested as further motivations beyond latency hiding for intermediate cache levels. These effects were not a focus of the *PicoServer* study. In this study, we consider whether these concerns might change the answer on whether or not to flatten the hierarcy. We consider data-centric and scientific workloads with larger memory footprints than the more network-centric benchmarks for which *PicoServer* was designed (the higher percentage of DMA and I/O accesses in web serving reduces L2 effectiveness). Additionally, we consider the impact of asymmetric read and write latencies, a characteristic of many non-volaltile memories like PCM [21, 22] and STT-RAM [24].

Our intent is to provide guidance on memory hierarchy design for single-chip servers, such as the *Nanostores* [23] proposal by HP Labs. *Nanostores* are single-chip servers with 3D (or 2.5D) stacked logic and dense non-volatile memory. *Nanostores* are further distinguished by their use of low-voltage throughput-oriented many-core processors. Overall, *Nanostores* have been proposed as viable building blocks for scalable data-centric systems.

Since the advent of 3D stacking with through-silicon vias (TSVs), there have been a number of studies to examine 3D stacked memory system design. Loh [16] describes a 3D memory organization that increases "memory level parallelism through a streamlined arrangement of L2 cache banks, MSHRs, memory controllers and the memory arrays themselves". His work improves page-level parallelism by increasing row buffer cache capacity, thereby allowing for a larger set of open memory pages. Loh also uses a data structure called the Vector Bloom Filter to reduce the number of probings required to determine hits and misses in the L2. Loh and Hill [17] explore efficient on-chip DRAM-based caching using conventional cacheline sizes. Their technique schedules tag and data accesses as compound accesses such that the data accesses are always row buffer hits, thereby making hits faster than just storing tags in the DRAM. Their technique also makes misses faster by using MissMaps to reduce stacked-DRAM accesses on misses. Woo et al. [27] propose SMART-3D, a new 3D-stacked memory architecture with a vertical L2 fetch and writeback network using a large array of TSVs. SMART-3D leverages TSV bandwidth to hide latency behind very large data transfers. All of these designs include an intermediate cache level between the L1 and stacked memory hierarchy level, and do not explore flattening the memory hierarchy in the 3D-stacked context.

Several groups have created prototypes of 3D stacked memory systems [6, 11, 28]. Most academic prototypes have omitted intermediate caches, largely due to area constraints rather than specific design optimization objectives.

Recent work on single-chip servers draws inspiration from earlier work on the *RAMpage* memory hierarchy proposed by Machanick and Salverda [19]. The *RAMpage* memory hierarchy uses DRAM as a paging device, and moves main memory up one level to the lowest level of SRAM. The main

motivation for the *RAMpage* hierarchy is to reduce the cost of DRAM references. While RAMpage does not eliminate the L2, it realizes a memory hierarchy that provides similar trade-offs to that proposed in *PicoServer*.

3. Methodology

We follow a trace-based methodology to evaluate the profiltability of eliminating intermediate cache levels; this methodology allows us to rapidly study workloads with large footprints. We collect memory traces with *PIN* [18] on a server system with eight 1.9 GHz Intel Xeon cores, 32 kB private L1 caches, an 8 MB 16-way set-associative shared L2 cache, and 16 GB of main memory. The traces contain information on the address, program counter, size, and type (read or write) of each memory access, and the number of non-memory instructions between consecutive memory accesses. We leverage performance counters to measure the peak L2 and memory bandwidth requirements of each workload on this server system using the Linux *Perf* performance analysis tool for use in our bandwidth filtering study.

To assess the impact of various memory subsystem designs, we replay L1 access traces through the *gem5* architectural simulator [4]. The trace replay emulates 1 GHz single-issue in-order cores. We replay one billion memory accesses per core. We collect statistics for the number of accesses to the L2 and stacked main memory; these statistics allow us to calculate the average L1 miss latency and L2/memory energies under various assumptions. When the L2 is enabled, we simulate an 8 MB 16-way associative shared L2 (i.e., the same L2 cache organization as in our actual x86 server).

As our goal is to identify technology inflection points, we do not select specific latency or static/dynamic energy-per-access values to represent particular memory technologies. Rather, we fix L2 accesses at 10 cycles and 1 nJ per access (which are in the typical range for current technology), and then vary the ratio of stacked memory read latency, write latency, and dynamic energy relative to these values. We also explore a wide range of L2 leakage assumptions.

We focus our study on a suite of four scientific and data-centric server applications. We include *Canneal* and *Fluidanimate* from the PARSEC Benchmark Suite [3] (using the native inputs), *Integer Sort* (*IntSort*) from the NAS Parallel Benchmark Suite, and *Graph500*. For comparison, we also study four SPEC CPU2006 benchmarks with differing (but known) locality characteristics. From SPEC, we include: *h264* and *sjeng*, which have comparitively small working sets, and *milc* and *soplex*, which have large working sets [1]. The SPEC applications are single-threaded and hence are recorded and replayed using only one core.

4. Bandwidth Filtering

Nearly three decades ago, Goodman first pointed out that caches can be used as bandwidth filters [7]. We first consider whether application bandwidth needs create a requirement for an intermediate cache between L1 and stacked memory. Using *Perf*, we assess the L2 and memory bandwidth requirements of each of our applications on an 8-core x86 server at one-second granularity. The L2 bandwidth consumption measured on our test system would correspond to the load offered to a stacked main memory if the L2 were elided.

Figure 1 and Figure 2 show the bandwidth utilization between the L2 cache and main memory, and between the L1 and L2 caches for the non-SPEC and SPEC benchmarks respectively. The SPEC results reflect the demands of a single Xeon-class core, while the non-SPEC results show the aggregate bandwidth demand of eight cores. As shown in the figures, the bandwidth between the L2 and main memory in the presence of an L2 is under 2,600 MB/s for all benchmarks. Without an L2 cache, the bandwidth required for all benchmarks aside from *IntSort* is under the peak bandwidth of 17 GB/s that one channel of DDR3 can provide [2]. *IntSort* requires a peak bandwidth of approximately 29 GB/s, which can be satisfied by two DDR3 channels. More importantly, this bandwidth requirement is several factors below the projected 192 GB/s available with 3D-stacked memory as projected by Woo et al. [27].

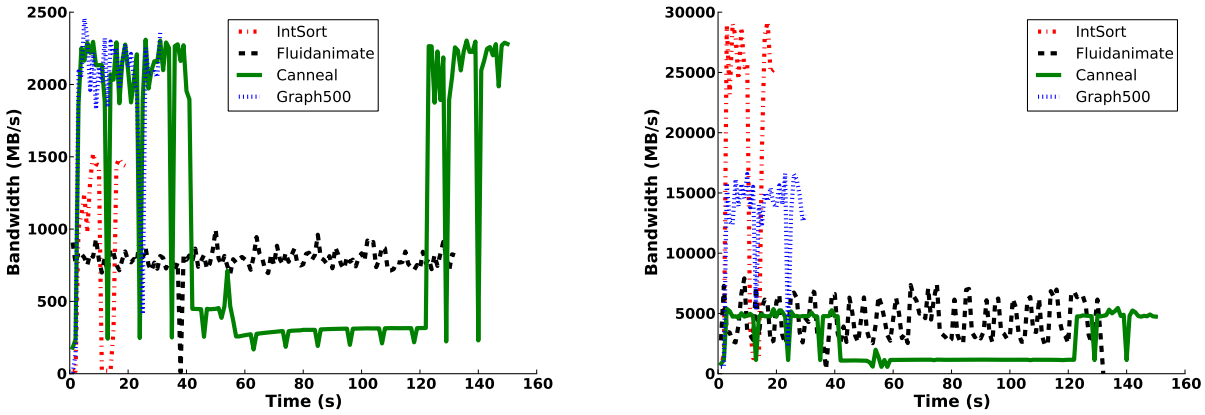
The Xeon cores used in our experiment are much more aggressive than those envisioned for *Nanostores*. Nevertheless, even for aggressive cores, it is clear that 3D-stacked memory provides ample bandwidth, and bandwidth filtering is moot for performance. Hence, we find that *PicoServer's* recommendation to flatten the hierarchy is viable from a bandwidth perspective. Note however, that bandwidth filtering may be valuable when stacking non-DRAM memory technologies like PCM that may suffer from limited endurance [13, 22]. As seen from Figure 1, eliding the L2 can increase traffic to the stacked memory by up to 20x.

5. Latency

We next consider the latency impact of intermediate caches. We evaluate two scenarios. In the first scenario, we assume that the memory read and write latencies are equal, representing stacked DRAM. In the second scenario, we consider the impact of scaling write latency relative to read latency, as slower writes are common in emerging memory technologies. In each scenario, we contrast systems with and without an L2. When enabled, we fix the L2 latency at 10 cycles and assume that L2 and memory are accessed in series. We vary the ratio of stacked memory to L2 access latency. We use average L1 miss latency as our evaluation metric.

5.1 Symmetric Read and Write Latencies

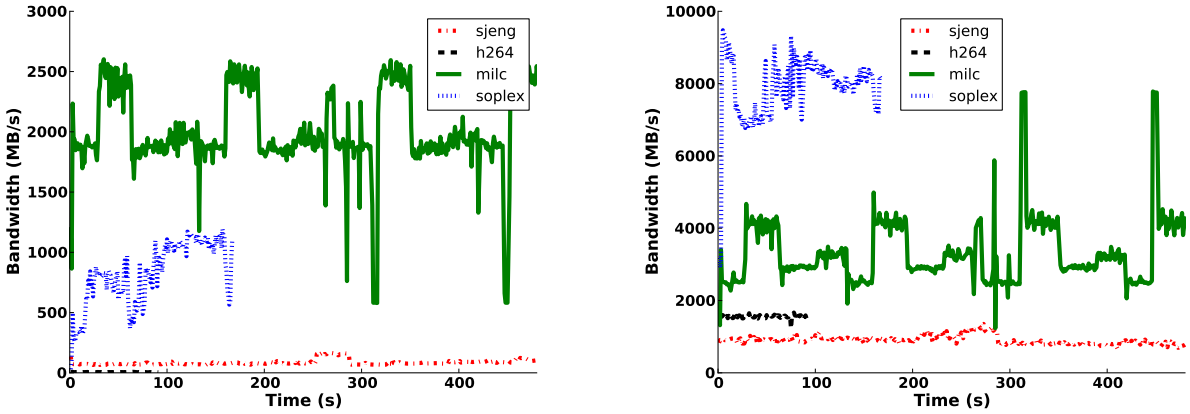
Figure 3 and Figure 4 show the average L1 miss latency for non-SPEC and SPEC benchmarks, respectively. The x-axes on the graphs represent the memory access latency normalized to the L2 access latency. The solid black line on each



(a) Main Memory Bandwidth

(b) Inter-cache Bandwidth

Figure 1: (a) Bandwidth between the L2 and main memory for the non-SPEC benchmarks: The highest bandwidth requirement is under 2,500 MB/s. (b) Bandwidth between the L1 and L2 caches for the non-SPEC benchmarks: This is the bandwidth between the L1 and main memory in the absence of an L2. The highest bandwidth requirement is about 29,000 MB/s.



(a) Main Memory Bandwidth

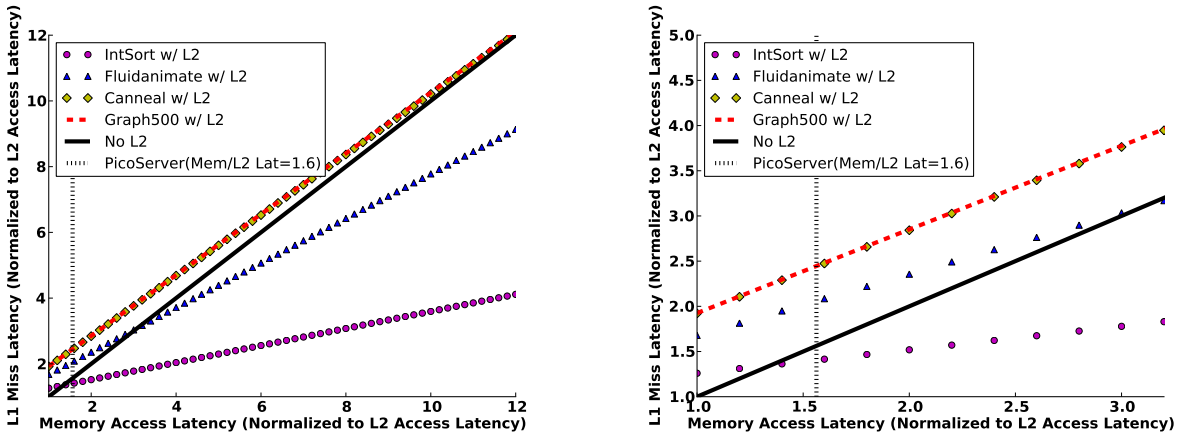
(b) Inter-cache Bandwidth

Figure 2: (a) Bandwidth between the L2 and main memory for the SPEC benchmarks: The highest bandwidth requirement is under 2,600 MB/s. (b) Bandwidth between the L1 and L2 caches for the SPEC benchmarks: This is the bandwidth between the L1 and main memory in the absence of an L2. The highest bandwidth requirement is about 9,500 MB/s.

graph with a slope of one displays the average L1 miss latency in the absence of an L2, in which case all L1 misses incur a main memory access (note that latency is therefore the same for all benchmarks). The remaining lines show the L1 miss latency for each benchmark; their origin and slope depends on the hit rate achieved by the 8MB L2. The dotted vertical line indicates the latency ratio assumed in the *PicoServer* study. Of particular interest is the point where each workload's line intersects the "No L2" line; this latency ratio represents the inflection point where an L2 becomes profitable. That is, for stacked memory latencies greater than this inflection point, the time saved on L2 hits outweighs the time lost on misses. This inflection point can be calculated directly given an L2 miss rate using the well-known average memory access latency formulas described by Hennessey and Patterson [8].

Among the non-SPEC benchmarks, only *IntSort* has a sufficient L2 hit rate to warrant an intermediate 8MB L2 for the likely range of stacked DRAM access latency ratios (in the vicinity of the *PicoServer* assumption of 1.6x). It breaks even at a ratio of 1.4x. *Fluidanimate* might derive some benefit if the stacked DRAM is quite distant from the L2, for example, if it is implemented with a slower memory technology. *Canneal* and *Graph500* have exceedingly poor temporal locality in the L2, and hardly benefit even when stacked memory has an access latency similar to off-chip DRAM. Our conclusion is that the *PicoServer* recommendation to flatten the hierarchy will typically hold for this application class.

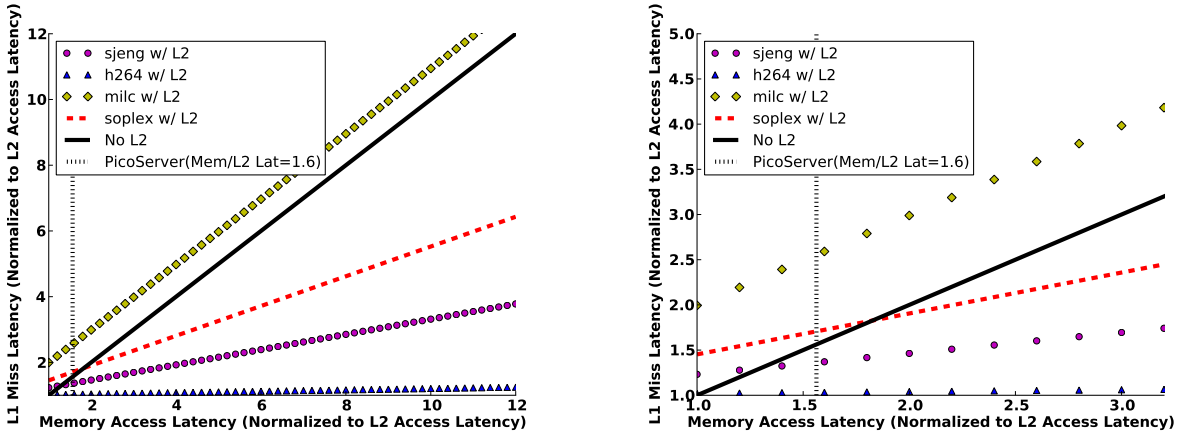
We notice a different behavior with the SPEC applications. As the majority of SPEC applications have much smaller



(a) Average L1 Miss Latency

(b) Average L1 Miss Latency

Figure 3: Average L1 miss latency for the non-SPEC benchmarks given symmetric memory read and write latencies. Memory latency is normalized to L2 access latency. As memory latency increases, *IntSort* and *Fluidanimate* benefit from an L2 for relatively memory low latencies. Performance of both *Canneal* and *Graph500* is similar with and without an L2 (a) *IntSort* and *Fluidanimate* benefit from having an L2 beyond a memory latency of 1.4x and 3.1x respectively. *Canneal* and *Graph500* only show improvement beyond a memory latency of 12.7x and 13.4x respectively. (b) Zoomed in graph to show detail.



(a) Average L1 Miss Latency

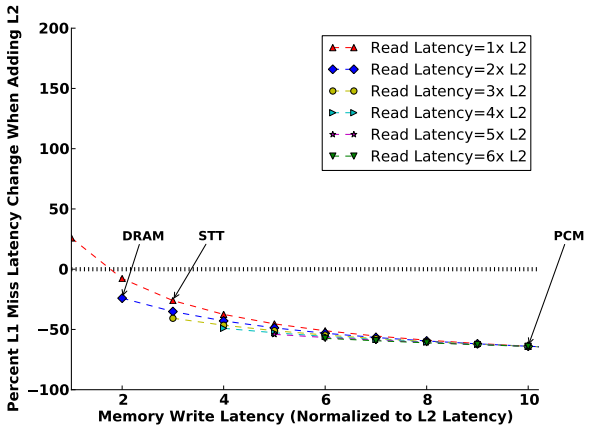
(b) Average L1 Miss Latency

Figure 4: Average L1 miss latency for the SPEC benchmarks given symmetric memory read and write latencies. Memory latency is normalized to L2 access latency. *sjeng*, *h264*, and *soplex* benefit from an L2. (a) *sjeng* and *soplex* benefit from having an L2 beyond a memory latency of 1.3x and 1.8x respectively. *h264* always benefits from an L2. *milc* does not benefit from an L2. (b) Zoomed in graph to show detail.

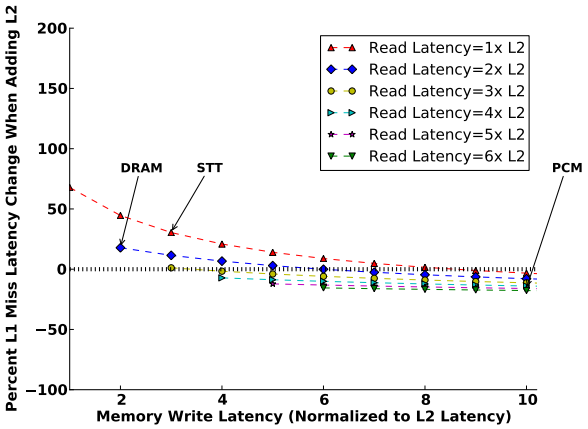
footprints and working sets than our data-centric workloads, they tend to benefit from a stacked L2 even at relatively low memory latency ratios. Three of our four selected apps (*h264*, *sjeng*, and *soplex*) likely benefit from an L2, only *milc* is better off without. Note that we selected *soplex* because it has one of the largest working sets among the SPEC applications, yet it still rarely spills out of the L2 compared to the data-centric workloads. Because they are designed to stress CPU rather than memory system performance, SPEC applications do not place significant pressure on the caches. A memory system optimized to run SPEC may perform poorly for scientific and data-centric applications.

5.2 Asymmetric Read and Write Latencies

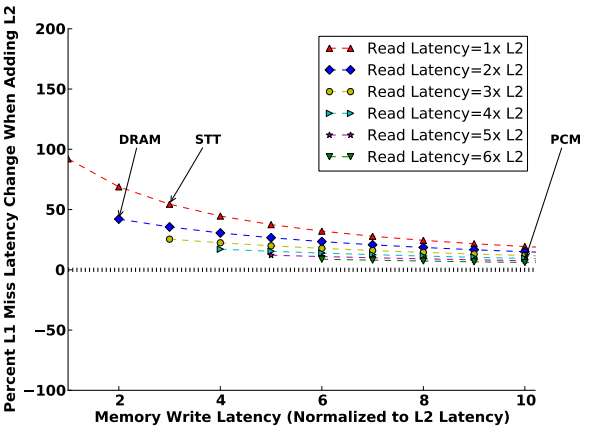
Figure 5 and Figure 6 show the percent change in average L1 miss latency when adding an L2 for the eight benchmarks assuming asymmetric memory read and write latencies (we only consider cases where the write latency is equal to or higher than the read latency). We normalize the memory read and write latencies to the L2 access latency. This experiment models the read-write latency asymmetry that is anticipated in emerging memory technologies, and examines the performance impact of coalescing L1 write-back traffic within the L2. The figures also indicate the approximate write-to-read latency ratio for a few emerging memory tech-



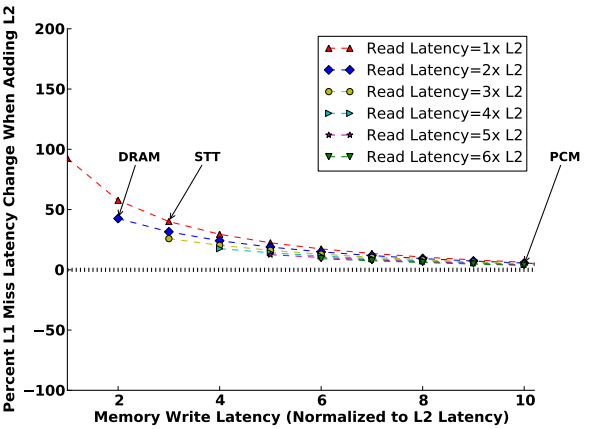
(a) Average L1 Miss Latency: *IntSort*



(b) Average L1 Miss Latency: *Fluidanimate*



(c) Average L1 Miss Latency: *Canneal*



(d) Average L1 Miss Latency: *Graph500*

Figure 5: Percent change in average L1 miss latency for the non-SPEC benchmarks under conditions of asymmetric memory read and write latencies: Memory read and write latency are normalized to L2 the access latency. Points below the dotted line (y=0) favor inclusion of an L2, while points above the dotted line favor exclusion of the L2. In general, an L2 is not beneficial except for *IntSort*. *Fluidanimate* finds an L2 beneficial only when read and write latencies are over 4x the L2 access latency.

nologies [9, 21, 24]. Points above the dotted horizontal line (y=0) on each graph indicate a reduction in average L1 miss latency when eliminating the L2, while points below the dotted line indicate a reduction in average L1 miss latency with the inclusion of an L2. In other words, points above the dotted horizontal line favor exclusion of an L2, while points below the line favor inclusion of an L2.

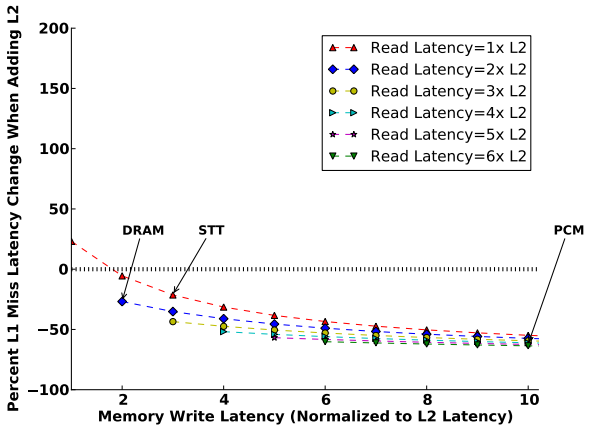
In all but one case of the SPEC workloads, slower writes degrade performance when the L2 is elided (*milc* sees no difference because the L2 is ineffective for this workload). However, among the non-SPEC applications, with the exception of *IntSort*, the performance gap is small. Hence, we conclude that for data-centric workloads, an intermediate cache is not warranted even if stacked memory writes are substantially slower than reads. Again, we see a markedly different behavior in SPEC apps, which have substantial L1 writeback traffic, and benefit from write coalescing in L2.

Table 1: Memory Access Latency Inflection Points

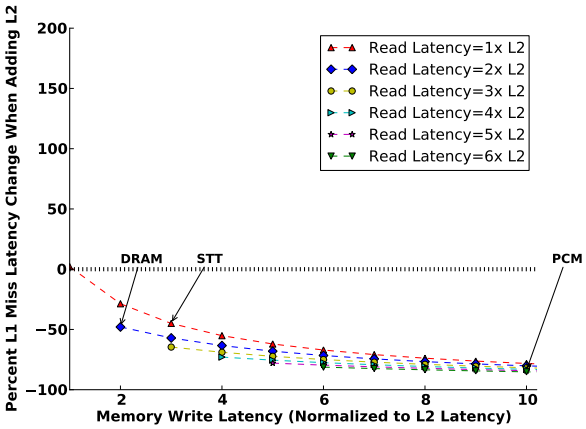
Benchmark	Latency
IntSort	1.4x
Fluidanimate	3.1x
Canneal	12.7x
Graph500	13.4x
sjeng	1.3x
h264	1.0x
milc	178.5x
soplex	1.8x

6. Energy Filtering

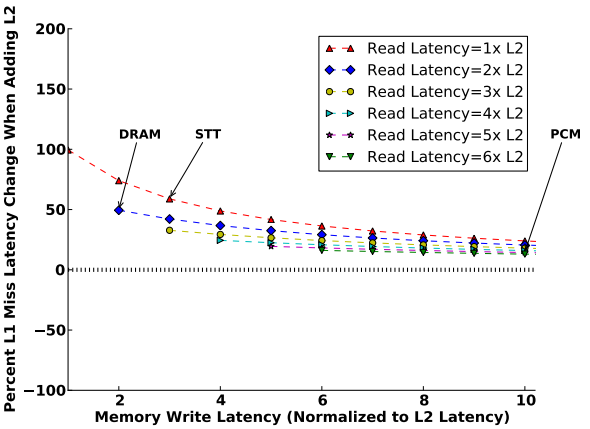
Finally, we consider the impact of an intermediate cache as an energy filter. To simplify our analysis, we select the stacked memory access latency on a per workload basis. The memory latency value is normalized to the L2 access latency



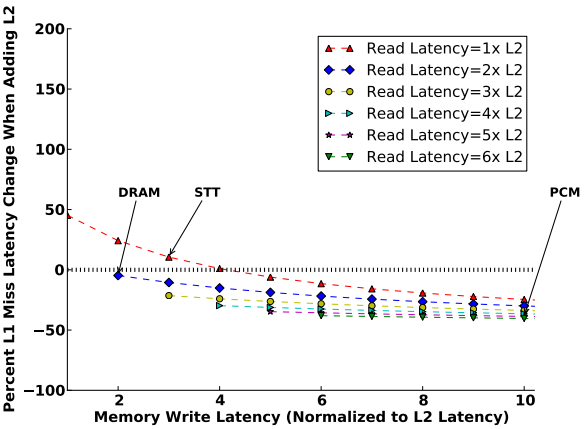
(a) Average L1 Miss Latency: *sjeng*



(b) Average L1 Miss Latency: *h264*



(c) Average L1 Miss Latency: *milc*



(d) Average L1 Miss Latency: *soplex*

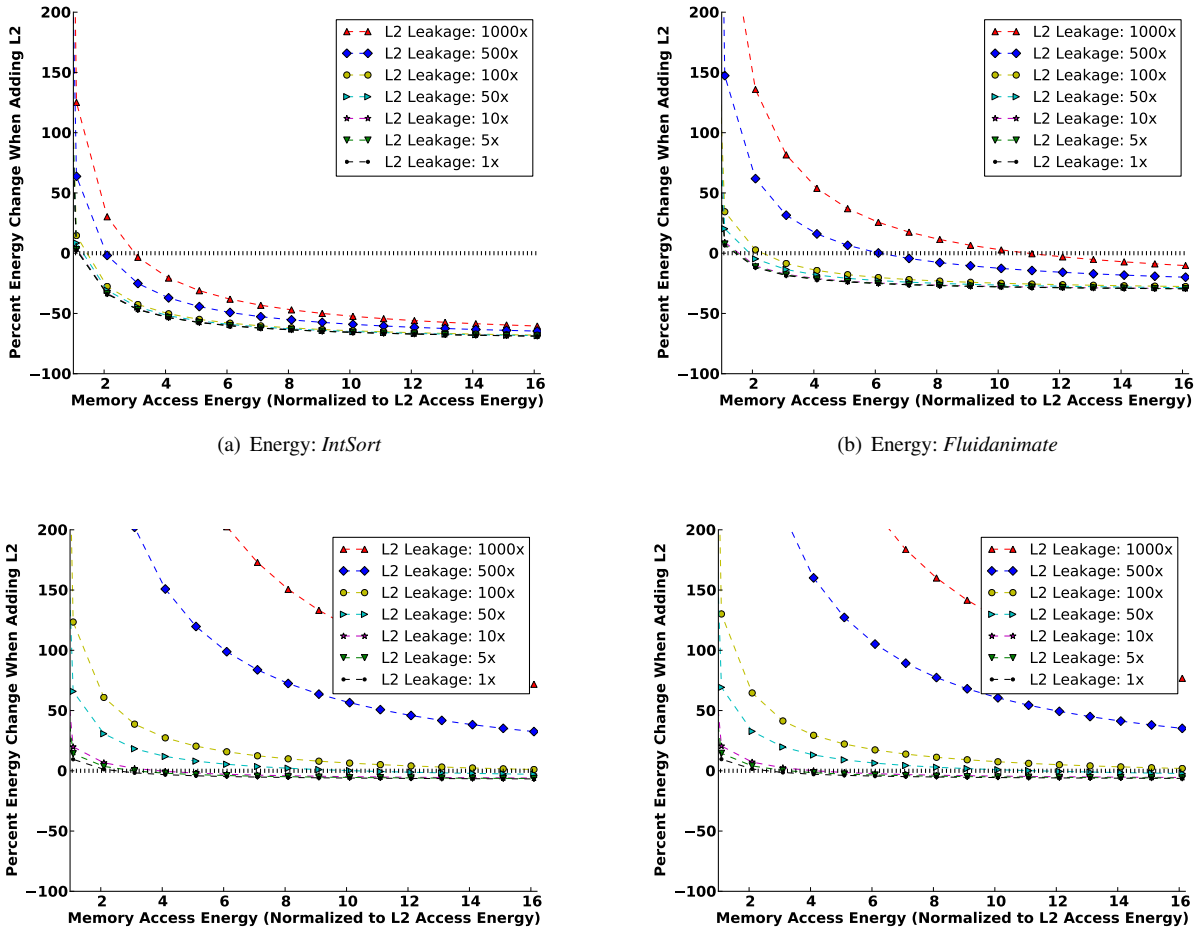
Figure 6: Percent change in average L1 miss latency for the SPEC benchmarks for asymmetric memory read and write latencies: Points below the dotted line (y=0) favor inclusion of an L2, while points above the dotted line favor exclusion of the L2. When memory writes are significantly slower than memory reads, an L2 improves performance for all benchmarks except *milc*.

such that the application runtime is equal with and without an L2. At this point, due to the equal runtimes, the leakage energy of the stacked memory is the same with and without an L2. Hence, we can neglect the stacked memory leakage in our analysis. Table 1 lists the memory latency values used for each workload.

We hold the L2 dynamic energy fixed at 1 nJ per access (a typical value for an 8MB L2 in current technology), and vary the stacked memory energy-per-access relative to this fixed value. The ratio of main memory to L2 dynamic energy is reflected on the x-axis in Figure 7 and Figure 8. We also vary the L2 static energy over a wide range, from 1 mW (1x leakage) to 1W (1000x leakage). We sweep leakage power over this large range because L2 leakage can vary by orders of magnitude due to two effects. First, leakage can vary relative to dynamic energy by a factor of ten based on the activity factor of the L2 cache. Second, leakage can vary by nearly 100x based on the particular circuit implementation selected for the cache (e.g., low-operating-power vs. high performance cells).

Figure 7 and Figure 8 show the percent energy change when adding an L2 for the non-SPEC and SPEC benchmarks, respectively. Points above the dotted horizontal line (y=0) on each graph indicate energy savings when eliminating the L2, while points below the dotted line indicate lower energy with inclusion of an L2.

Because DRAM accesses require relatively little dynamic energy, it is likely that the memory-to-L2 access energy ratio will be quite low (below 1.5) for stacked DRAM (stacked DRAM accesses are projected to require 1 to 1.5 nJ, similar to the dynamic energy of an access to an 8MB SRAM). At this ratio, *none* of the applications benefit from an L2 from an energy filtering perspective. Even if stacked memory dynamic energy-per-access turns out to be substantially worse than expected (e.g., greater than 4x the L2 dynamic energy-per-access), L2 will still only be profitable if L2 leakage en-



(a) Energy: *IntSort*

(b) Energy: *Fluidanimate*

(c) Energy: *Canneal*

(d) Energy: *Graph500*

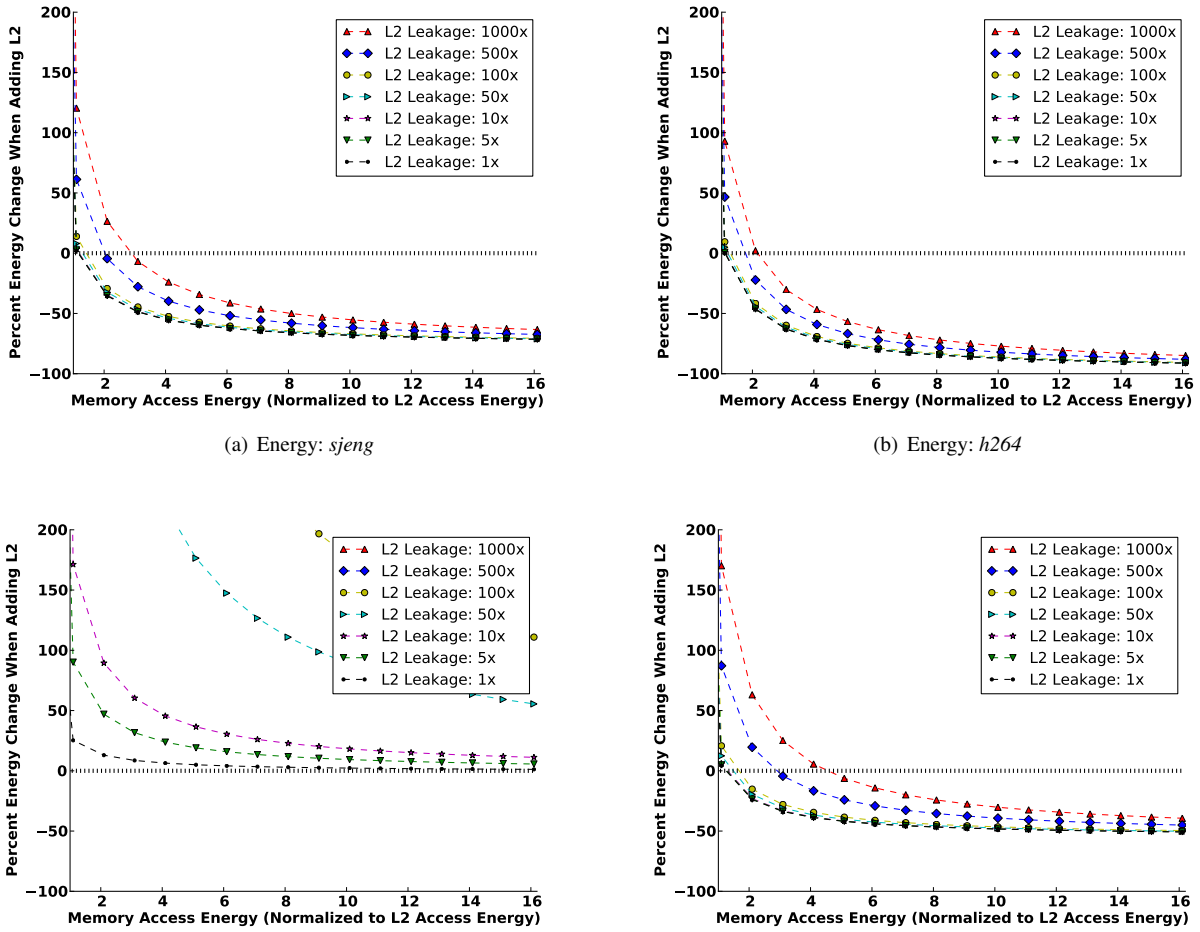
Figure 7: Percent energy change when adding an L2 for the non-SPEC benchmarks: Memory access energy is normalized to L2 access energy. L2 leakage is swept from a baseline of 1x (1 mW) up to 1000x (1W). Points below the dotted line (y=0) favor inclusion of an L2, while points above the dotted line favor exclusion of the L2. (a) *IntSort* benefits from having an L2 beyond a memory access latency of 4x over the baseline, regardless of leakage. (b) *Fluidanimate* benefits from having an L2 beyond a memory access latency of 12x over the baseline, regardless of leakage. For lower memory access energy, and higher leakage, *Fluidanimate* does not require an L2. (c) and (d) *Canneal* and *Graph500* do not require an L2 in general.

ergy is low. For example, for the data centric applications, L2 leakage must be below 100 mW to provide a substantial energy benefit for any application except *IntSort*. Hence, we conclude that, from an energy perspective, an intermediate cache is unlikely to provide an energy advantage; the intermediate cache conserves energy only if (1) the L2 hit rate is high, (2) the stacked memory requires at least 4x the energy per access of the L2, and (3) the L2 leakage energy is low (well below 100mW). Given the good energy characteristics projected for stacked DRAM, we find it unlikely that all three of these conditions will typically hold.

7. Conclusion

Recent years have witnessed the emergence of 3D die stacking which has made single-chip server designs feasible. Moreover, memory technology is poised to undergo a transformation with the advent of non-volatile memories. 3D stacking alters key assumptions of conventional memory hierarchy design by drastically increasing cross-layer bandwidth and reducing the latency ratio between successive layers. In light of these technology trends, the *PicoServer* study has suggested flattening the memory hierarchy and eliminating caches that lengthen the critical path between the L1 and stacked memory.

In this paper, we revisit the conclusions made by *PicoServer*. By considering various design factors like the latency and energy ratios between the L2 and main memory, along with various L2 leakage values, we identify inflection points where an L2 becomes profitable. We guide our investigation by analyzing intermediate caches from three perspectives:



(a) Energy: *sjeng*

(b) Energy: *h264*

(c) Energy: *milc*

(d) Energy: *soplex*

Figure 8: Percent energy change when adding an L2 for the SPEC benchmarks: Memory access energy is normalized to L2 access energy. L2 leakage is swept from a baseline of 1x (1 mW) up to 1000x (1W). Points below the dotted line (y=0) favor inclusion of an L2, while points above the dotted line favor exclusion of the L2. (a) and (b) *sjeng* and *h264* benefit from having an L2 beyond a memory access latency of 3x over the baseline, regardless of leakage. (c) *milc* never benefits from the inclusion of an L2. (d) *soplex* benefits from having an L2 when the memory access latency exceeds the baseline by over 5x.

bandwidth, *latency*, and *energy filtering*. We focus our study on data-centric and scientific workloads and draw contrasts with the SPEC applications that are frequently used in computer architecture studies.

We conclude that bandwidth filtering is moot given the plentiful bandwidth provided by 3D stacking. From a performance (latency) perspective, although a flat hierarchy is optimal for access patterns with poor temporal locality, some workloads benefit from the access latency reduction afforded by an L2. This effect is more prominent when memory write and read latencies are asymmetric, a characteristic of memory technologies like PCM and STT-RAM. Finally, from an energy perspective, we find that for scientific and data-centric workloads with large footprints, an L2 cache is unlikely to be beneficial from an energy perspective. In particular, we find that low L2 leakage power is critical. SPEC applications tend to have good hit rates in an 8MB cache and hence favor inclusion of an L2; we conclude that these benchmarks should be used with caution when designing memory hierarchies for data-centric applications.

In summary, we find that the conclusions drawn in the *PicoServer* study hold true for the data-centric and scientific workloads. For these benchmarks, having an L2 is only profitable under the following situations: (1) when the L2 leakage is small, (2) when main memory is relatively distant compared to the L2, and (3) for write-intensive workloads when memory write latency exceeds memory read latency.

Acknowledgements

This work was partially supported by NSF CSR-0834403 and US Dept. of Energy Award DE-SC0005026 (please note disclaimer at http://www.hpl.hp.com/DoE-Disclaimer.html).

References

[1] SPEC CPU2006 Memory Characterization. http://www.jaleels.org/ajaleel/workload.

[2] Samsung Electronics: Samsung High-performance SDRAM for Main Memory. 2007.

[3] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, 2011.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 Simulator. *SIGARCH Computer Architecture News*, 39(2).

[5] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, et al. Die Stacking (3D) Microarchitecture. In *Proc. 39th Annual Intnl. Symp. on Microarchitecture*, 2006.

[6] D. Fick, R. G. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, et al. Centip3De: A 3930DMIPS/W Configurable Near-threshold 3D Stacked System with 64 ARM Cortex-M3 Cores. In *Intnl. Solid-State Circuits Conf.*, 2012.

[7] J. R. Goodman. Using Cache Memory to Reduce Processor-Memory Traffic. In *Proc. 10th Ann. Intnl. Symp. on Computer Architecture*, 1983.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2006.

[9] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers. Improving Write Operations in MLC Phase Change Memory. In *Proc. 18th Intnl. Symp. on High Performance Computer Architecture*, 2012.

[10] 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 *Proc. 12th Intnl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2006.

[11] D. H. Kim, K. Athikulwongse, M. Healy, M. Hossain, M. Jung, I. Khorosh, G. Kumar, Y.-J. Lee, D. Lewis, T.-W. Lin, et al. 3D-MAPS: 3D Massively Parallel Processor with Stacked Memory. In *Proc. Intnl. Solid-State Circuits Conf.*, 2012.

[12] J. Kin, M. Gupta, and W. Mangione-Smith. The Filter Cache: An Energy Efficient Memory Structure. In *Proc. 30th Ann. Intnl. Symp. on Microarchitecture*, 1997.

[13] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory As a Scalable DRAM Alternative. In *Proc. 36th Ann. Intnl. Symp. on Computer Architecture*, 2009.

[14] D. Li, J. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu. Identifying Opportunities for Byte-Addressable Non-Volatile Memory in Extreme-Scale Scientific Applications. In *Proc. Intnl. Parallel and Distributed Processing Symp.*, 2012.

[15] 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), 2005.

[16] G. H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *Proc. 35th Ann. Intnl. Symp. on Computer Architecture*, 2008.

[17] G. H. Loh and M. D. Hill. Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches. In *Proc. 44th Ann. Intnl. Symp. on Microarchitecture*, 2011.

[18] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Janapa, and R. K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Programming Language Design and Implementation*, 2005.

[19] P. Machanick and P. Salverda. Preliminary Investigation of the RAMpage Memory Hierarchy. *South African Computer Journal*, (21):16 –25, August 1998.

[20] K. Puttaswamy and G. Loh. 3D-Integrated SRAM Components for High-Performance Microprocessors. *IEEE Trans. Computers*, 58(10), 2009.

[21] M. Qureshi, M. Franceschini, and L. Lastras-Montano. Improving Read Performance of Phase Change Memories Via Write Cancellation and Write Pausing. In *Proc. 16th Intnl. Symp. on High Performance Computer Architecture*, 2010.

[22] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *Proc. 36th Ann. Intnl. Symp. on Computer Architecture*, 2009.

[23] P. Ranganathan. From Microprocessors to Nanostores: Rethinking Data-Centric Systems. *Computer*, 44(1):39 –48, January 2011.

[24] M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili. An Energy Efficient Cache Design Using Spin Torque Transfer (STT) RAM. In *Intnl. Symp. on Low-Power Electronics and Design*, 2010.

[25] W. Tang, R. Gupta, and A. Nicolau. Design of a Predictive Filter Cache for Energy Savings in High Performance Processor Architectures. In *Intnl. Conf. on Computer Design*, 2001.

[26] S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell. Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory. In *Proc. 9th USENIX Conf. on File and Storage Technologies*, 2011.

[27] D. H. Woo, N. H. Seong, D. Lewis, and H.-H. Lee. An Optimized 3D-stacked Memory Architecture by Exploiting Excessive, High-density TSV Bandwidth. In *16th Intnl. Symp. on High Performance Computer Architecture*, 2010.

[28] M. Wordeman, J. Silberman, G. Maier, and M. Scheuermann. A 3D System Prototype of an eDRAM Cache Stacked over Processor-like Logic Using Through-Silicon Vias. In *Intnl. Solid-State Circuits Conf.*, 2012.