Exploring Cache Size and Core Count Tradeoffs in Systems with Reduced Memory Access Latency

Paulo C. Santos, Marco A. Z. Alves, Matthias Diener, Luigi Carro, Philippe O. A. Navaux
Informatics Institute – Federal University of Rio Grande do Sul – Porto Alegre, Brazil
Email: {pcssjunior, mazalves, mdiener, carro, navaux}@inf.ufrgs.br

***Abstract*—One of the main challenges for computer architects is how to hide the high average memory access latency from the processor. In this context, Hybrid Memory Cubes (HMCs) can provide substantial energy and bandwidth improvements compared to traditional memory organizations. However, it is not clear how this reduced average memory access latency will impact the LLC. For applications with high cache miss ratios, the latency to search for the data inside the cache memory will impact negatively on the performance. The importance of this overhead depends on the memory access latency. In this paper, we present an evaluation of the L3 cache importance on a high performance processor using HMC also exploring chip area tradeoffs between the cache size and number of processor cores. We show that the high bandwidth provided by HMC memories can eliminate the need for L3 caches, removing hardware and making room for more processing power. Our evaluations show that performance increased 37% and the EDP improved 12% while maintaining the same original chip area in a wide range of parallel applications, when compared to DDR3 memories.**

Keywords*—*Cache memories, HMC, chip area tradeoff

I. Introduction

Multi-core architectures are a reality in modern embedded and high performance systems. With the advance of manufacturing techniques, the number of processors is likely to increase from the current tens of cores to several hundreds of cores or more [1]. Following this trend, we can predict that the pressure on the memory sub-system will keep on increasing. For this reason, the well-known memory wall problem [2], [3] has been growing in modern parallel computers.

Different solutions have emerged over the last years that aim to circumvent the traditional DDR memories limits, such as the low bandwidth and high latency. The increase on the number of channels and memory controllers aims to better expose the bank parallelism, reducing thus the average memory access latency. Such techniques require more data buses to interconnect the memory controllers to the multiple memory modules. On the other hand, the processor architecture also included bigger cache memories with complex hierarchies in order to hide the memory access latency from the cores. Thus, data requests from the processor will trigger cache searches, which cost a certain number of cycles. During this search, in case the data is present in the cache, the request is serviced. In case the data is not located in the cache, the request needs to be forwarded to the next memory level, which can be another cache level or the main memory. Such data requests normally are performed sequentially between the levels in order avoid unnecessary searches increasing the energy efficiency.

However, considering the new Hybrid Memory Cube (HMC) memories [4], it is not clear if the ever increasing cache memories still benefit the system performance. The HMC memories promise high data bandwidths at a low average latency due to their aggressive parallelism [5]. According to the latest designs available, this new memory architecture is formed by 32 vaults, each consisting of 8 or 16 DRAM banks [4], [6]. The memory banks are split into multiple stacks and connected to the logic layer using Through-Silicon Vias (TSVs) [7]. The logic layer provides computational capabilities (atomic update instructions) and also substitutes the memory controller from the processor, as it is responsible for handling the requests and for sending the DRAM signals to the banks.

The memory footprint of applications is continuously increasing [8]. However, most of the data is reused only far in the future, or not reused at all. Therefore, the large cache memories closer to the main memory may not improve the performance. The reason is that on every cache hit, the cache memories help to improve the performance, but during cache misses, the extra cycles required to search for the data inside the cache ends up hurting the performance. For high latency main memories, the benefits of large caches hide this small overhead. However, if the main memory has a reduced latency, the cache benefits may not be worth the extra overhead.

In this scenario, we revisit the cache memories in this paper to understand the impact of the HMC memories on the cache hierarchy. We focus on the L3 cache, due to its large area usage and high energy consumption. We explore the area tradeoffs between L3 cache and number of cores per chip, in order to evaluate the performance impact of exchanging the cache size for a larger core count. To the best of our knowledge, our work is the first to explore the HMC benefits for the cache memory subsystem, analyzing the tradeoffs between the cache size and the core counts inside the chip.

In our simulations, we found that HMC-based systems can reduce the average memory access latency by 65% compared to DDR3 memories with 4 channels. This latency reduction using HMC memories enabled us to remove the L3 cache from the system, losing on average only 7% of performance. When removing this cache and adding more cores maintaining the same chip area, we observed an average speedup of 37% while decreasing the Energy Delay Product (EDP) by 12%.

II. Impact of HMC on Cache Memories

The memory footprints of applications increases as the complexity of the problems being solved rises [9]. Although most applications reuse data along their execution, in case the data is reused after the cache already evicted the respective

978-1-4673-8776-7/16 $31.00 © 2016 IEEE
DOI 10.1109/PDP.2016.55

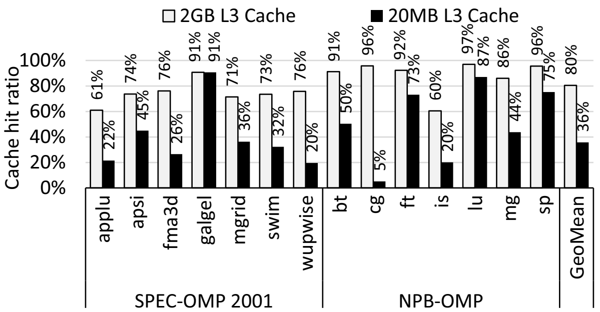

Fig. 1: L3 cache hit ratio comparing two hypothetical caches.

data, this reuse will not benefit from the cache and instead will suffer a penalty due to higher latency from accessing multiple cache levels before the main memory.

A. Impact of the L3 Cache Size on HMC

To understand the memory access behavior of the applications present in traditional parallel benchmark suites, Figure 1 presents the L3 hit ratio with different cache sizes. The experiment is based on a Sandy Bridge machine with 8 cores and compares a realistic multi-banked 20 MB L3 cache to a hypothetical 2 GB L3 that is able to accommodate the memory footprint of all applications. In both systems, the L1 and L2 caches remain at their original parameters and HMC is used as the main memory. The simulation and the workload details will be presented in Section IV.

The results with 2 GB L3 cache show that on average, cache lines on L3 receive between 5 and 6 accesses during the program execution, resulting in a cache hit ratio of 80%. It means that even using a huge cache memory, on average 20% of the L3 cache requests require main memory accesses due to a cold cache effect. For the 20 MB L3 cache, on average the cache lines received less than 2 accesses, where the first caused the cache miss and the second represents the reuse, achieving a hit ratio of only 36%. These results demonstrate that for a realistic cache hierarchy, more than half of the L3 requests require main memory accesses, and for those cache misses, the cache search represents only an overhead before the main memory services the request.

The influence of the cache search latency overhead on the final performance depends mainly on two factors, the number of cache accesses with the respective cache miss ratio and also the main memory latency. Considering a cache with a high number of accesses and a high miss ratio, the impact on the performance tends to be high. On the other hand, if the cache search latency is much smaller than the main memory latency or the cache have a low number of accesses, the impact on the performance will be low.

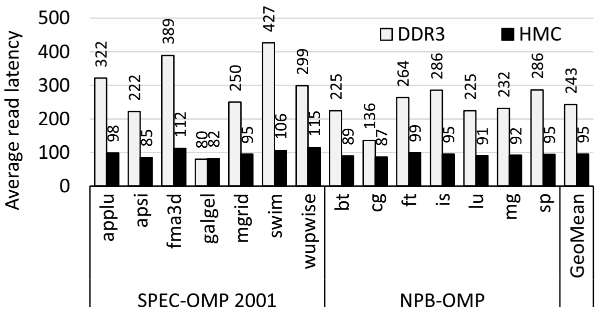

Fig. 2: Average ready latency for DDR3 and HMC memories.

B. Comparing HMC and DDR3 Memories

To understand the access latency differences between DDR3 and HMC memories, Figure 2 presents the average number of processor cycles during main memory accesses. The results are based on two systems formed by 8 cores and 20 MB L3 cache. The first with DDR3 memory and 4 channels, the second with HMC and 4 links.

The results show that HMC, with its higher level parallelism between the multiple vaults and banks, reduced the average memory read latency by 60%. This means that systems with DDR3 memories requires a low cache miss ratio due to the high latency and small bandwidth of the main memory. However, for the new HMC memories, the tradeoff between the gains provided by the cache hits and the losses caused by the extra cache latency during the misses has changed.

Considering that the cache memories can occupy up to half of chip area on current parallel high performance processors [10], the area tradeoffs need to be analyzed in order to understand if the area could be better used to obtain a higher performance. Changes in the main memory architecture have a direct impact on the cache subsystem performance, and thus, we propose to evaluate architectural modifications, exploring the cache memory size and the core count on the chip.

III. Related Work

The gap between the processing and memory performance is a well-known issue [2]. In order to hide the high latency and low bandwidth of the SDR and DDR memories, different techniques were proposed. Bigger cache memories and more complex memory hierarchies [11], [12] are being adopted by the industry as a way to deliver a higher bandwidth with a reduced latency per access. However, such big cache sizes occupy large areas of the chip, while also requiring more complex interconnections and more attention of coherence mechanisms as the number of levels increases.

Despite the high density and low power consumption potentials of hybrid cache memories [13], [14], [15], which use emerging technologies such as FeRAM, Memristors or flash memories together with traditional memories (SRAM and DRAM), these new techniques are still not much more than a promise, mainly due to their lower write endurance, lower performance and multiple technological integration challenges.

Multi-channel DDR [16] or FBDIMM memories [17] are also used to offer higher bandwidth by increasing the bank and module parallelism. However, multi-channel DDR requires wider interconnections which increase their energy consumption, while still not supporting multi-core memory pressure [3]. Moreover, the bandwidth can be increased further by adding multiple memory controllers, which requires a more complex interconnection design and also occupies large chip areas.

In order to reduce the memory access latency, SRAM caches can be inserted into the DRAM memories [18], [19], storing reusable data pages in these caches. The main limitation of such techniques is the reduced cache size, which is related to the larger cell size of SRAMs compared to DRAMs. Their data consistency and the extra cache latency are also issues that need to be considered during the design.

Multiple studies evaluated the implementation impact and details of off-chip DRAM cache memories [20], [21]. Such cache memories reside outside the processor chip and can hold large amounts of data due to the high integration capabilities of DRAM cells. However, the latency of such systems depends on the DRAM performance and also on the tag store implementation. Although previous work has evaluated the performance of HMC memories compared to the standard DDR modules, no studies have explored the impact of this new memory organization on the cache memory sub-system, which can expose new tradeoffs present on future systems and support radical architectural changes.

IV. Simulation Methodology

To understand the possible tradeoffs between the cache hit benefits and the extra latency during cache misses, we executed a variety of simulation experiments. This section presents the proposed architectures to be evaluated, explaining the design decisions, the simulation methodology, and also the parallel benchmarks suites.

A. *Proposed Architectural Designs*

The baseline architecture for our experiments is inspired by the Sandy Bridge processor with 8 cores and 20 MB L3, as detailed in the next subsection.

Figure 3 presents the percentage of the chip area used by each component of the baseline system. We observe that the L3 occupies 29% of the chip area while each core with its respective L1 and L2 caches represents less than 7%. Based on this information, we present three possible scenarios (A, B and C) for using the area relative to the L3 cache. The possibility A represents the unchanged baseline. For the possibility B, we plan to save the L3 cache area and evaluate the performance of such system. Finally for the third possibility (C), we design a processor with 4 extra cores with their respective L1 and L2 cache, occupying the same area of the original L3 cache.

B. *Simulation Environment and Benchmark Suites*

To evaluate the different architectures, we used a cycle-accurate simulator [22]. We used the McPAT [10] modeling tool, version 1.3, in order to calculate the energy usage and chip area of each component of the system, enabling us to perform comparisons regarding the chip area tradeoffs. The parameters of the simulated system are shown in Table I.

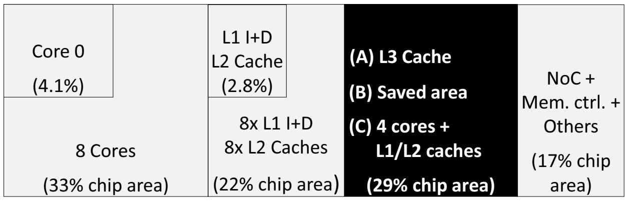

Fig. 3: Baseline system with L3 cache area possibilities.

TABLE I: Simulation parameters for Sandy Bridge.

Processor Cores: 8 cores @ 2.0 GHz, 32 nm; 4-wide out-of-order; 16 stages
16 B fetch size; 18-entry fetch buffer, 28-entry decode buffer; 168-entry ROB;
MOB entries: 64-read, 36-write; 3-alu, 1-mul. and 1-div. int. units (1-3-32 cycle);
1-alu, 1-mul. and 1-div. fp. units (3-5-10 cycle); 1-load and 1-store units (1-1 cycle);
Branch Predictor: 1 branch per fetch; 4 K-entry 4-way set-assoc., LRU policy BTB;
48-entry BOB; Two-Level GAs 2-bits; 16 K-entry PBHT; 256 lines, 2048 sets SPHT;

L1 Data + Inst. Cache: 32 KB, 8-way LRU, 64 B line size; 2-cycle;
MSHR: 8-request, 10-write-back, 1-prefetch; Stride prefetch: 1-degree, 16-strides;

L2 Cache: Private 256 KB, 8-way LRU, 64 B line size; 4-cycle;
MSHR: 4-request, 6-write-back, 2-prefetch; Stream prefetch: 2-degree, 256-streams;

L3 Cache: Shared 20 MB (8-banks), 20-way LRU; 64 B line size; 6-cycle;
Bi-directional ring; MOESI coherence protocol; MSHR: 8-request, 12-write-back;

DDR3-1333 Modules: On-core ctrl.; 4 GB; DRAM@166 MHz; 8 KB row buffer
8 DRAM banks; 4-channels; 8 B burst width at 2:1 core-to-bus freq. ratio;
Open-row policy; DRAM: CAS, RP, RCD, RAS and CWD latency 9-9-9-24-7 cycles;

HMC v2.0 Module: 4 GB; 32 vaults, 8 DRAM banks/vault; 256 B row buffer;
DRAM@166 MHz; 4-links@8 GHz; 4 B burst width at 2:1 core-to-bus freq. ratio;
Closed-row policy; DRAM: CAS, RP, RCD, RAS and CWD latency 9-9-9-24-7 cycles;

We experimented with 14 parallel benchmarks from two suites: 7 applications from SPEC-OMP2001 [23], and 7 applications from NAS-NPB [24]. All benchmarks were compiled for x86-64 using GCC with the `-O3` flag. The SPEC-OMP2001 benchmarks were executed using the *reference* input, while the NAS-NBP suite uses input size *A*. Each benchmark executes up to one time step from its parallel region.

V. Experimental Evaluation

This section presents our experimental results, starting with a comparison of the DDR3 and HMC memories. Then, we remove the L3 cache of the system and show memory-related statistics to understand the system performance. Finally, we present an evaluation of the tradeoffs involved in increasing the core count while removing the L3 cache.

A. *DDR3 and HMC Results*

Figure 4 presents the speedup, comparing our baseline system using HMC with 4 links to a system using DDR3 with 4 channels. We notice that in all applications that we evaluated, the HMC memory is able to at least match the performance presented by DDR3, achieving average gains of 90%.

We can observe that *galgel* presented no performance difference (speedup of 1.00), which is explained by the very high L3 cache hit ratio achieved (91%) and the memory footprint (less than 5 MB) smaller than the L3 cache size. This means that few memory accesses are handled by the main memory. On the other hand, the *swim* application has a low L3 cache hit ratio (31%) and a big memory footprint (1.4 GB), and therefore takes more advantage of the low average read latency provided by HMC memories, which results in a substantial speedup compared to DDR. We can also observe that although the *mg* and the *mgrid* applications implement the same algorithm, due to their internal parameters, such as the grid size and the grid sparsity applied, these applications present different behaviors and thus different performance results.

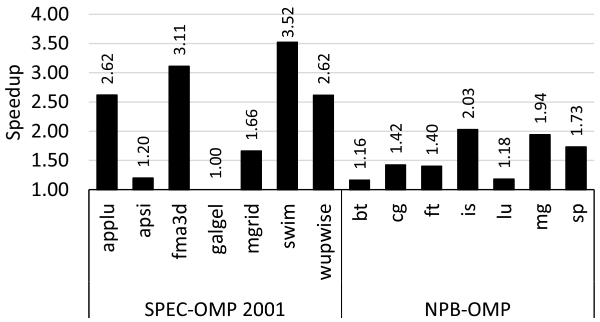

Fig. 4: HMC speedup over DDR3 memory.

B. Impact of the L3 Cache on Performance

Starting the analysis regarding the L3 cache influence when HMC memory is used, Figure 5 shows the speedup results comparing a system with only L1-L2 caches, to the baseline with L1-L2-L3 caches. Removing the L3 cache leaded to an average performance degradation of 4%, where speedup values <1 indicate that the L3 cache improves the system performance. It is possible to observe that applications with high data reuse (and by consequence more L3 cache hits, as shown in Figure 1), the performance is directly dependent on the L3 cache.

Besides the cache hit ratio, another relevant metric to understand the speedup results is the Misses per Kilo Instructions (MPKI), which represents the pressure that the L2 cache puts on the next memory level, that can be the L3 cache or the main memory. Figure 6 L2 cache MPKI of the evaluated benchmarks. It shows that the average pressure that the L3 cache or the main memory needs to service is 27.8 accesses per kilo instructions.

In our experiments we could observe that applications with a L2 MPKI above 10 and a cache hit ratio lower than 35% matches with the those applications (*applu*, *fma3d*, *swim*, *wupwise*, *cg* and *is*) that improved the performance by removing the L3 cache. For applications with L2 MPKI greater or equal than 10 and hit ratio between 36% and 45% (*mgrid* and *mg*), only small performance degradations occurred (less than 5%).

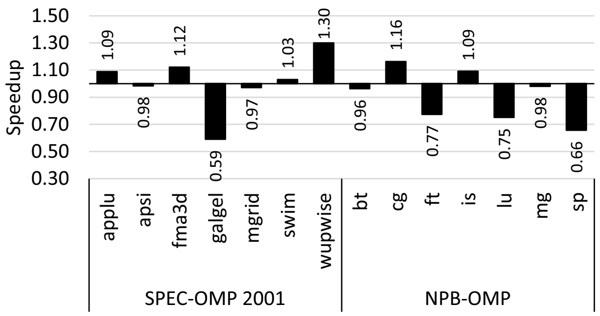

Fig. 5: Speedup of an 8-core HMC system without an L3 cache normalized to a system with 20 MB of L3 cache.

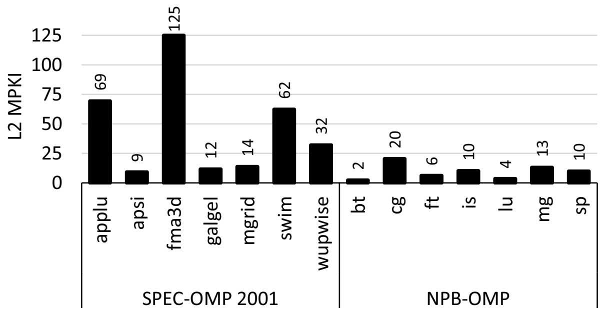

Fig. 6: L2 cache pressure over the next memory level (L3 cache or main memory).

C. Chip Area Tradeoff Results

Considering that the LLC can occupy a large chip area, this section presents a design space exploration considering different numbers of cores while area usually occupied by the L3 cache is freed. Figure 7 presents performance, power, area, and EDP for different systems using HMC without the L3 cache, varying the number of cores between 8 and 16. The results are normalized to the baseline system with 8 cores, 20 MB of L3 cache and HMC.

These results show that when more cores are implemented in the system, more pressure is applied on the main memory, making a better use of the HMC's internal parallelism (vaults and banks). The best result is achieved with 16 cores, due to more processing elements available requiring more pressure and the HMC memory that is able to serve this pressure. In this case, the average speedup of 1.56 over the baseline is achieved with an area overhead of 20% and a power overhead of 52%. Despite the overhead, the EDP was reduced by 35%, showing that energy efficiency can be increased. When limiting the chip area to the baseline, we can place up to 12 cores in the system by removing the L3 cache. Such a configuration presents a speedup of 1.20, an area reduction of 10% and an EDP reduction of 12%.

In Figure 8, we present an additional experiment, executing with 12 cores in three different configurations: 12 cores without L3 cache, 12 cores with 10 MB L3 cache and 12 cores with the full 20 MB L3 cache. The results show that for most benchmarks, increasing the L3 only improves

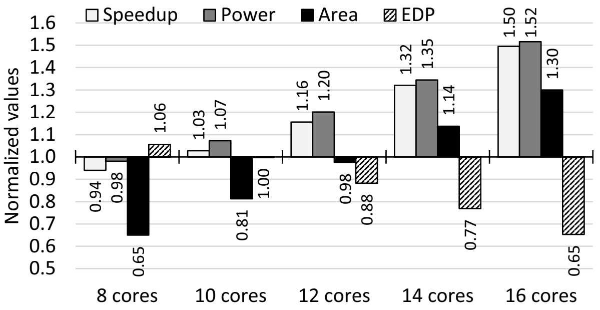

Fig. 7: Average results of different HMC systems without L3 cache compared to a system with 8 cores and 20 MB L3.

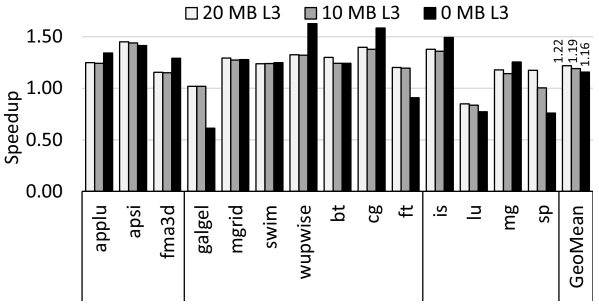

Fig. 8: Comparing an HMC system with 12-cores and different amounts of L3 cache, normalized to the 8-cores + 20 MB L3.

performance slightly. Only three applications, *galgel*, *ft*, and *sp*, profit significantly from the larger cache. As before, several benchmarks reduce performance with larger caches due to the extra memory access latency.

VI. Conclusions and Future Work

During the last years, high performance processors started to provide an increasing number of cores per chip. Similarly, their cache memories kept on increasing and started to consume vast areas of the chip. Such cache memories exploit the spatial and temporal locality of memory accesses in order to hide the high latency and small bandwidth of the traditional DDR memories from the processor. With the new HMC memories sustaining the performance even under high memory pressure, while fewer memory accesses can benefit from large caches due to their low temporal locality, the tradeoff between cache size and number of cores can be explored. Our research evaluates this tradeoff by comparing different hardware systems, exploring the implementation of different cache levels and numbers of cores.

In our experiments, we found that applications with a low temporal or spatial locality, with L3 cache hit ratio lower than 45%, benefits or have a low impact on the performance when removing the L3 cache (maximum 3% of performance reduction). By removing the L3 cache from our baseline system we observed on average an 4% performance degradation. However, the area of this big cache could be better utilized by adding more processing capabilities. Our proposed architecture with 12 cores provided performance improvements of 16% with EDP reduction of 12%, using the same baseline chip area and increasing the overall efficiency.

Acknowledgments

We thank CNPq and CAPES for their support. This work was developed within the HPC4E project of RNP/EU.

References

[1] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, “Toward dark silicon in servers,” *IEEE Micro*, vol. 31, pp. 6–15, 2011.

[2] D. Burger, J. Goodman, and A. Kagi, “Limited bandwidth to affect processor design,” *IEEE Micro*, vol. 17, no. 6, pp. 55–62, Nov 1997.

[3] Z. Zhu and Z. Zhang, “A performance comparison of DRAM memory system optimizations for SMT processors,” in *Int. Symp. on High-Performance Computer Architecture*, 2005.

[4] Hybrid Memory Cube Consortium, “Hybrid Memory Cube Specification Rev. 2.0,” 2013, http://www.hybridmemorycube.org/.

[5] K. Khalifa, H. Fawzy, S. El-Ashry, and K. Salah, “Memory controller architectures: A comparative study,” in *Int. Design and Test Symp.*, 2013.

[6] J. Jeddeloh and B. Keeth, “Hybrid memory cube new DRAM architecture increases density and performance,” in *Symp. on VLSI Technology*, 2012.

[7] J. V. Olmen, A. Mercha, G. Katti *et al.*, “3D stacked IC demonstration using a through Silicon Via First approach,” in *Int. Electron Devices Meeting*, 2008.

[8] R. Murphy and P. Kogge, “On the memory access patterns of supercomputer applications: Benchmark selection and its implications,” *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 937–945, July 2007.

[9] S. Biswas, B. de Supinski, M. Schulz, D. Franklin, T. Sherwood, and F. Chong, “Exploiting data similarity to reduce memory footprints,” in *Int. Parallel Distributed Processing Symp.*, 2011.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing,” *ACM Transactions on Architecture and Code Optimization*, vol. 10, 2013.

[11] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts, “A fully integrated multi-CPU, GPU and memory controller 32nm processor,” in *Int. Solid-State Circuits Conference Digest of Technical Papers*, 2011.

[12] J. M. Borkenhagen, R. D. Hoover, and K. M. Valk, “Exa cache/scalability controllers,” *IBM Enterprise X-Architecture Technology: Reaching the Summit*, pp. 37–50, 2002.

[13] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, “Power and performance of read-write aware hybrid caches with non-volatile memories,” in *Design, Automation Test in Europe Conference Exhibition*, 2009.

[14] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, “Hybrid cache architecture with disparate memory technologies,” in *Int. Symp. on Computer Architecture*, 2009.

[15] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, “Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design,” in *Design, Automation Test in Europe Conference Exhibition*, 2012.

[16] Z. Zhu and Z. Zhang, “A performance comparison of DRAM memory system optimizations for SMT processors,” in *Int. Symp. on High-Performance Computer Architecture*, 2005.

[17] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, “Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling,” in *Int. Symp. on High Performance Computer Architecture*, 2007.

[18] Z. Zhang, Z. Zhu, and X. Zhang, “Cached DRAM for ILP processor memory access latency reduction,” *IEEE Micro*, no. 4, pp. 22–32, 2001.

[19] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian, “CHOP: Adaptive filter-based DRAM caching for CMP server platforms,” in *Int. Symp. on High Performance Computer Architecture*, 2010.

[20] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, “Exploring DRAM cache architectures for CMP server platforms,” in *Int. Conf. on Computer Design*, 2007.

[21] Z. Zhang, Z. Zhu, and X. Zhang, “Design and optimization of large size and low overhead off-chip caches,” *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 843–855, 2004.

[22] M. A. Z. Alves, M. Diener, F. B. Moreira, C. Villavieja, and P. O. A. Navaux, “SiNUCA: A Validated Micro-Architecture Simulator,” in *High Performance Computation Conference*, 2015.

[23] H. Saito, G. Gaertner, W. Jones, R. Eigenmann, H. Iwashita, R. Lieberman, M. van Waveren, and B. Whitney, “Large system performance of SPEC OMP2001 benchmarks,” in *Int. Symp. on High Performance Computing*, 2006.

[24] H. Jin, M. Frumkin, and J. Yan, “The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance,” Tech. Rep., 1999.