Cache Utilization as a Locality Metric: A Case study on the Mantevo Suite
(CSCI-ISPD, Full/Regular Research Paper)

Nafiul Alam Siddique*, Patricia Grubel*, Abdel-Hameed A. Badawy*, and Jeanine Cook†
*Klipsch School of Electrical and Computer Engineering, New Mexico State University,
P.O. Box 30001, MSC 3-O, Las Cruces, NM 88003-8001
†Sandia National Labs, Albuquerque, NM

***Abstract*—Cache hierarchies have long been utilized to minimize the latency of main memory accesses by caching frequently used data closer to the processor. Significant research has been done to identify the most crucial metrics of cache performance. Though the majority of research focuses on measuring cache hit rates and data movement as the major cache performance metrics, cache utilization can be equally important. In this work, we present cache utilization performance metrics that provide insight into application behavior. We define cache utilization in two forms: 1) the fraction of data bytes in a cache line that are actually accessed at least once before eviction from cache and 2) the access frequency of data bytes in a cache line. We discuss the relationship between the utilization measurement and two important application properties: 1) spatial locality - the use of data located near data that has already been accessed, and 2) temporal locality - or the reuse of data over time. In addition to measuring cache line utilization performance, we present conventional performance metrics as well to illustrate a holistic understanding of cache behavior. To facilitate this work, we build a memory simulator incorporated into the Structural Simulation Toolkit (SST). We measure and analyze the performance for several scientific mini-applications from the Mantevo suite [1]. This work justifies that caches are not necessarily the best on-chip solution for all types of applications due to its fixed line size.**

Keywords – Cache Utilization, Locality, Workload Characterization, Cacheline Utilization, Mantevo Applications, Multicore Cache Simulation

I. Introduction

The memory system is one of the main bottlenecks for processor performance due to the memory wall [2]. Caches can alleviate the problem but not solve it. Cache stores frequently used data near the processor. Typically, cache exploits locality of reference exhibited by all applications. There are two types of locality of reference: 1) spatial locality, which is the use of data items in memory near recently used items (locality in space), and 2) temporal locality, which is the reuse of a data item over time (locality of time). However, though the performance of an application in cache is predominantly dependent on how it exploits the locality of reference, there is no direct method to determine locality.

Typically, the performance of cache can be explained by performance metrics such as cache miss rate, data movement between memory hierarchies, data access latency and power consumption etc. Besides these performance metrics, another metric, *cache utilization* can be used to demonstrate an application's behavior in cache. In this work, we define cache utilization as a combination of two utilization metrics: 1) cache line utilization that describes the fraction of a cache line (or number of bytes in a line) being accessed before the cache line gets evicted, and 2) byte utilization, defined as the frequency of accesses of a byte in a cache line. Here, line utilization and byte utilization describe spatial locality and temporal locality respectively. However, poor cache utilization implies that the application is wasting both cache space and memory bandwidth. Even though the idea of cache line utilization has been explored in literature [3]–[6] to design energy efficient caches with variable cache line sizes the methodology and the purpose of studying cache utilization in this work are different. Furthermore to our knowledge, there is no other prior work that has looked at byte utilization as a cache performance metric.

Among multiple types of applications, we focus on scientific applications to study the utility of both cache and byte utilization. Typically, scientific applications are compute intensive. Such applications normally fully utilize the processor computer power as long as the cache hierarchy and underlying memory system can support the application requirements adequately to exploit spatial and temporal locality. We use only four benchmarks,CoMD, HPCCG, miniFE, and miniMD, from the Mantevo [1] suite for space limitations. Our objective in this paper is to identify the locality of these applications through measuring both cache and byte utilization.

Cache line utilization has been primarily used to determine the variable cache line size on memory hierarchy of specific architectures to design an energy efficient cache for similar kinds of benchmarks. However, we believe that cache utilization can be applied in a wider context. It can: 1) Interpret other cache performance metrics such as data movement and miss rate, 2) Estimate spatial locality, and 3) Estimate temporal locality.

The rest of the paper is organized as follows: Section III discusses cache performance metrics and application properties, Section IV discusses methodology and experimental setup, Section V discusses our results, and Section II and

VI discuss the related works and conclusions respectively.

II. Related Work

Most of the related work presents only memory performance metrics such as cache miss rates obtained from hardware performance counters or simulation events to understand the application performance on a specific simulator [7]–[10]. Some approaches measure processor dependent metrics to find the similarity between benchmarks [11]–[13]. However, Conway et al. [13] measure performance for AMD Opteron processor but do not explore cache utilization as a performance metric. Studies on cache line utilization have proposed utilizing variable cache line size to improve utilization and minimize power consumption [4]–[6], [14], [15].

Computer architects have proposed several techniques to improve the utilization of cache blocks However, such techniques increase overhead, or access time. Among them Kumar et al. [4] who did an extensive study on cache line utilization for L1 and L2 caches and tracked cache lines in word granularity. Srinivasan [6] tracked cache utilization in cache line granularity and found that most of the benchmarks utilize less than 70% of a cache line. Both Srinivasan and Kumar estimated power consumption with CACTI. These studies do not relate data movement or miss rate to cache utilization.

Alkohlani et al. [16] did a thorough analysis of temporal and spatial locality for both Spec and the Mantevo benchmarks on a single core. Our study on cache utilization expands that portion of their study to include cache utilization and locality for multicores.

III. Cache Performance Metrics & Application Properties

In this section, we describe cache utilization as a performance metric and its relationship to the application inherent characteristics, spatial and temporal locality.

A. Cache Performance Metrics

Cache is small, fast memory which attempts to store a sub-set of frequently used data in a fixed size data block (cache line). However, the size of cache is small compared to main memory. This introduces the requirement of having to evict/replace some cache line(s) from the cache. The replacement policy, implemented in hardware, controls which cache lines are to be replaced. Therefore, it is inevitable that all the bytes of a cache line might not get accessed before the line is replaced, possibly resulting in inefficient use of the cache line [17]. This phenomena can be assessed using the performance metrics that we describe in this section.

1) Cache Utilization: Cache utilization can help determine the efficiency of using cache lines during the span of execution of an application. It can also be used to estimate the proportion of power wasted in the cache due to storing and transferring the unused portions of cache lines. We define two types of cache utilization: 1) *cache line utilization* - the fraction of data in a cache line that is accessed at least once before the cache line is evicted from cache and 2) *cache byte utilization* - the access frequency of bytes of data in a cache line.

The computed cache line utilization (CLU) is the percentage of cache line bytes that are touched, *i.e.* read or written, per cache line, divided by the cache line size (64 bytes in our experiments), and averaged across the number of cache lines in the cache, as shown in Equation 1 (where TB_n is the number of bytes touched in cache line *n* and *N* is the total number of cache lines).

$$CLU = \frac{\sum_{n=1}^{N} TB_n}{N * 64} \div 100 \tag{1}$$

The unused data items in a cache line represent potential savings in terms of used bandwidth and power consumed to bring, store, and evict those unused data item in each cache line for the entire cache. This is one major source of power wastage in caches.

2) Data Miss Rate: Miss rate is defined as the fraction of memory references not found in cache, Equation 2. If a memory reference is not found in cache, it is brought from higher levels of the memory system (other cache levels or main memory). The new cache line will replace another that the replacement policy predicts will not be used in the near future [17].

$$\textit{Miss Rate} = \textit{Cache Misses} \div \textit{References} \tag{2}$$

3) Data Movement: Data movement is the movement of data between levels of the cache hierarchy. Typically, when a cache miss occurs and replacing a cache line is necessary, dirty cache lines (*i.e.* cache lines with modified data) from lower level caches are written back to higher levels and are also included in data movement measurements. In this research, we only show the data movement between L1 and L2 cache.

B. Application Characteristics

Among many properties, we discuss the two most important application properties; spatial locality and temporal locality.

1) Spatial Locality (locality in space): Spatial locality is defined as the property of accessing nearby data to recently accessed data. In other words, if we access a data item in a certain location, spatial locality states that we are to access spatially close by data items in the near future. Therefore, if many of the bytes in a cache line are accessed at least once before the cache line is evicted from the cache, this cache line exhibits spatial locality. Thus, the metric cache line utilization describes spatial locality. Furthermore, if most of the bytes of a cache line are accessed at least once, then the access pattern of the application exhibits significant spatial locality [17].

2) Temporal Locality (locality in time): Temporal locality is defined as the property of accessing recently accessed data again in the near future. In other words, if we access a data item at time t, then temporal locality states that we are to access that same data item in the near future. Therefore, if a byte or a group of consecutive bytes (data) in a cache line is accessed multiple times before the cache line is evicted,

the data exhibit temporal locality. Thus, byte utilization corresponds to temporal locality. Moreover, if most of the bytes of an application are accessed multiple times, then the access pattern of the application exhibits significant temporal locality [17].

3) Working Set: We define the working set of an application as the summation of pages of an application accessed during its entire execution. In our experiment, we define the page size as 4 KBytes. However, this definition is partially different than conventional definition of working set, as conventional working set defines the summation of pages during a specific time interval [17].

IV. Experimental Setup

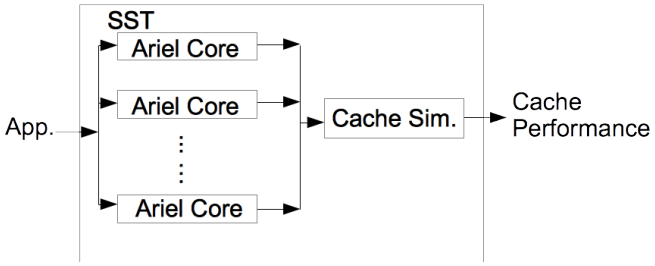

Fig. 1: Simulator Block Diagram

The hardware model of the cache hierarchy is implemented in the Structural Simulation Toolkit (SST) [18]. We implement the cache functional simulator in Ariel, a processor framework that can be generated in SST. In this framework, the application executes on multiple Ariel cores in the simulator environment, Figure 1. Memory references incurred by the application for each core of the Ariel processor model are fetched using PIN [19]. By design, PIN is integrated in Ariel.

A. Benchmarks

The Mantevo mini-apps [1], developed by Sandia National Laboratories, represent classes of scientific applications. The following descriptions are for four mini-apps we use to characterize their cache behavior:

CoMD is an example of a typical classical molecular dynamics algorithm and workload featuring the Lennard-Jones potential and the Embedded Atom Method potential.

HPCCG generates a synthetic linear system that generates a 27-point finite difference matrix in a sparse iterative format.

miniFE assembles a sparse linear-system from the steady-state conduction equation on a brick-shaped problem domain of linear 8-node hex elements and solves the linear-system using a simple un-preconditioned conjugate-gradient algorithm. After the end, the solution gets compared with the analytic solution of the problem.

miniMD is a typical molecular dynamics application that computes the force.

B. Cache Configuration

We implement a cache model with similar base configuration to the cache hierarchy of the Intel Sandy Bridge processor. The cache configuration is described in Table I.

TABLE I: Memory Configuration

L1D Private	32 KByte 4-way associative
L2 Private	256 KByte 8-way associative Non-inclusive
L3 Shared	8 MByte 16-way associative Inclusive

V. Cache Performance & Analysis

In this section, we present the performance results we have collected for cache utilization, miss rate and data movement for a set of applications from the Mantevo suite running in a multicore system. From the data we draw conclusions about the application properties. In our experiments, we run the applications with 1, 2, 4, 8, and 16 threads on 1, 2, 4, 8 and 16 cores respectively.

A. Cache Utilization

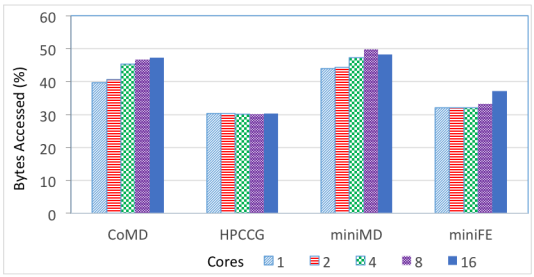

Fig. 2: Cache line utilization: average bytes accessed in the cache lines x

Figure 2, describes the average cache line utilization for all the benchmarks running on 1 – 16 cores. Almost for all the cases, cache line utilization increases with the core count since the cumulative size of caches increases proportionally with number of cores. Moreover, in multicore systems each thread executes on its own local data which can improve utilization. HPCCG exhibits the lowest utilization while miniMD shows the best utilization.

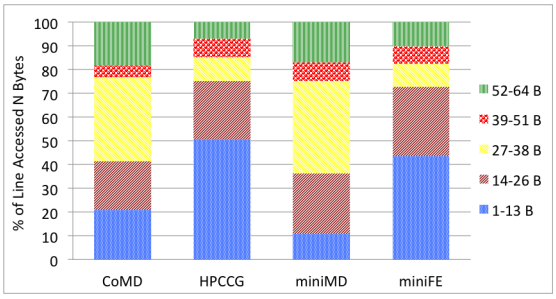

Fig. 3: Cache line utilization (8 cores): the percentage of cache lines that have accessed N bytes before eviction

The distribution of utilization of a cache line has been grouped into five bins, as is shown in Figure 3, where the

first bin contains the distribution of cache lines that between 1 and 13 Bytes of the cache line have been touched before the line is evicted. Similarly, the second, third, fourth, and fifth bins contain distribution of cache lines with 14-26 bytes, 27-38 bytes, 39-51 bytes and 52-64 bytes of data that have been touched respectively before the corresponding cache line is evicted from the cache. This cache line utilization histogram is collected while the benchmarks are running on 8 cores only. The y-axis represents the distribution of cache lines for the bins described above. We see that HPCCG performs poorly *i.e.* shows low levels of cache utilization. About 50% of the total cache lines falls in the 1 to 13 bytes bin. On the other hand, miniMD exhibits the best cache line utilization. About 62% of the total cache lines are in the bins with 27 or more bytes touched, followed closely by CoMD with approximately 59%. In addition, CoMD and miniMD show utilization for the last bin (52 bytes) of about 20% of the cache lines.

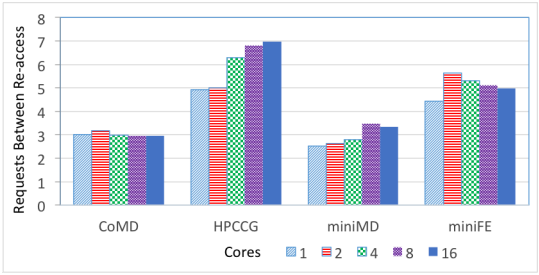

Fig. 4: Cache line reuse distance: the number of accesses between accesses to the same cache line

However, the cause of the variability in cache line utilization between these applications can be explained by Figure 4. In this figure, the y-axis presents the average number of memory requests between accesses to the same cache line (reuse distance). For HPCCG, the maximum reuse distance is between (4.5 to 6.5) which indicates that the cache line(s) can get evicted from the cache before its reuse (re-access) which leads directly to poor utilization (as the L1 cache is 4-way set associative). The pattern of cache line reuse distance varies for each applications with the number of cores.

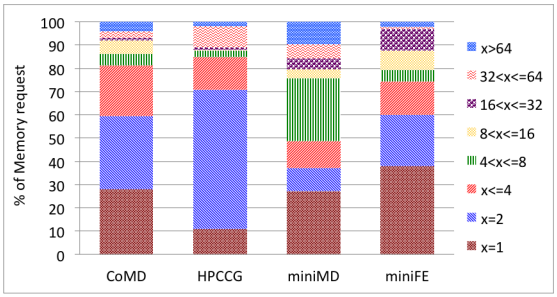

Fig. 5: Cache line reuse distance for 8 cores: the percentage of accesses that have x requests between accesses to the same cache line

The average cache line reuse distance for the applications running on 8 cores is shown in histogram format in Figure 5. HPCCG shows only 10% of the memory requests re-accesses to the same cache line, while the other applications range from 28 to 38%. Those applications also have a distance of two or less for about 60 to 70% of the memory requests.

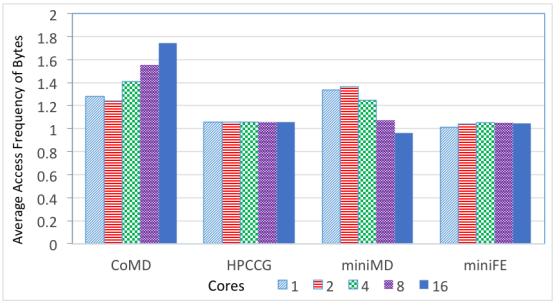

Fig. 6: Average access frequency of bytes

Figure 6 presents average count of accesses of data bytes per cache line. CoMD exhibits the maximum average number of accesses per data byte. Figure 7 presents byte reuse distance which is the average number of unique memory requests between reuses of the same data byte. The reuse distance of the same data byte for HPCCG is the largest compared to all other benchmarks, whereas CoMD exhibits the smallest distance.

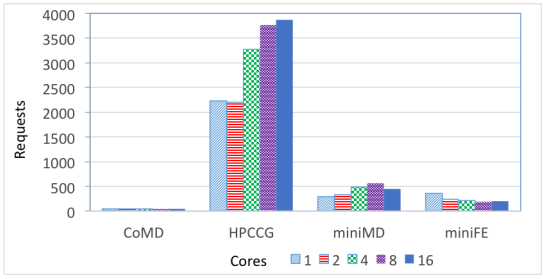

Fig. 7: Data bytes reuse distance: the number of requests between re-accesses to the same data byte

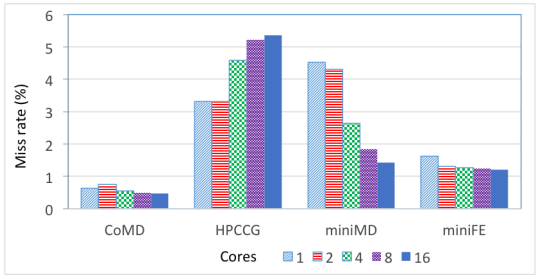

Fig. 8: Cache miss rate

B. Cache Miss Rate and Data Movement

Figures 8 and 9 show the cache miss rate and the average data movement for the different core counts respectively. The data movement is measured by summing the total data movement between L1 and L2 caches in both directions. These results suggest that due to the cumulative size of caches in a

multicore system, the cache miss rate is reduced with increased core count. However, due to coherence misses, data can get evicted from private caches which would increase the miss rate. HPCCG shows increased miss rate as the core count increases while all other applications exhibit reduced miss rates as core count increases.

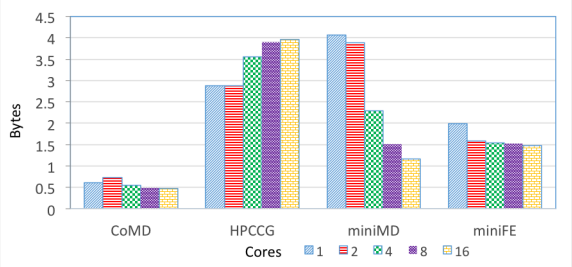

Fig. 9: Data movement (per access): the average number of bytes moved between the L1 and L2 caches

C. Application Properties

In this subsection, we investigate the inherent properties of the applications that influence the cache performance and describe them with the results presented in the previous subsection(s). Furthermore, we estimate the degree of locality from the performance data and measure the working set size with our simulation infrastructure.

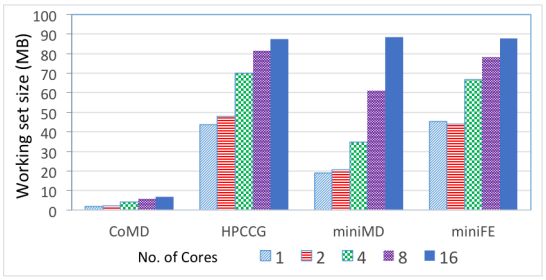

Fig. 10: Working set size

Figure 10 describes the working set size of the applications running on multiple cores. It is evident that the working set size increases with the number of cores for all applications. However, our experimental results show that CoMD has the smallest working set and miniFE has the largest working set. This goes hand in hand with the best miss rates and lowest data movement for CoMD. Also, it is consistent with the maximum byte access frequency and least byte reuse distance for CoMD (Figure 6 and 7 respectively). However, although HPCCG has a similarly sized working set as miniFE, but it shows a much larger byte reuse distance than miniFE. Typically, higher line and byte reuse distances are the causes of a higher miss rate as well as high data movement. The results also shows that miniMD extends its working set about 3.23 times while running on 8 cores compared to running on single core whereas HPCCG extends it by about 1.86 times only. Typically, local and temporary variables have private copies for each thread in virtual address space which increases the size of the working set. On the contrary, static and global variables share the same virtual address space with multiple threads. Therefore, most likely HPCCG has more static and global variables that are shared between threads which performs poorly (*i.e.* higher miss rates) when the application is multi-threaded.

The locality of an application can be explained by its cache (data) utilization. Figure 2 presents the cache line utilization. The cache lines in HPCCG are poorly utilized and miniMD exhibits the best utilization. From, this observation we can conclude that HPCCG exhibits poor spatial locality and miniMD on the contrary exhibits the best spatial locality for 64 byte cache lines. Moreover, in Figure 6, we show the average byte access frequency that depicts the temporal locality of the applications. From this figure we can conclude that HPCCG exhibits the lowest degree of temporal locality as the average number of access per byte of data is low (about 1). However, miniMD exhibits better temporal locality than HPCCG for 64 byte cache lines and CoMD exhibits the best temporal locality.

We clearly concluded from the studied benchmarks that some applications suffer low utilization at the cache line and byte levels. We conduct a small experiment to show that a cache with a variable cache line size [4] or a scratch pad memory [20] that is hardware managed with some limited compiler support can improve the performance and power consumption.

We build a 64 KB data structure that functions as a buffer that stores variable sized data blocks according to the size of the data sequences. Initially, we store each of the unique memory addresses in the buffer and add the consecutive memory addresses of the requests that have neighboring memory addresses that are already in the buffer. This tunes to the spatial locality of the application(s). We keep the buffer size equal to the L1 cache size so that we restrict keeping the data blocks in the buffer. Thus, we remove the data blocks when a new unique request appears which does not have neighbors stored in the buffer and the buffer is full.

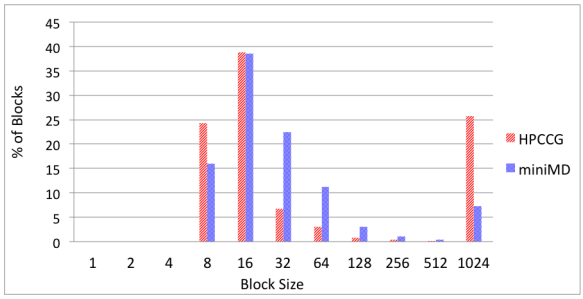

Fig. 11: Distribution of Block Sizes

Figure 11 shows the distribution of the data block counts with block sizes for HPCCG and miniMD running on a single core. For HPCCG, the size of about 73% of the data blocks is 64 bytes or less, whereas the size of 88% of the data blocks

is less than or equal to 64 bytes for miniMD. This can be one reason why miniMD performs better than HPCCG with a cache line of 64 bytes.

VI. Conclusions & Future Work

This work introduces cache utilization as a performance metric to estimate locality and to compare the data movement and miss rates between different applications. We report results on some of the benchmarks in the Mantevo benchmarks suite and find that the Mantevo benchmarks exhibit diverse behavior in all dimensions that prescribes different memory architecture for different applications. To our knowledge, this work is the first to present cache utilization as a performance metric to estimate locality and compare performance for running the Mantevo benchmarks in parallel.

We conclude from our results that a regular cache with a fixed cache line size might not be the best option for some applications. We believe that a variable line size cache or a scratch pad memory that is hardware-managed with some compiler support can attain good gains by avoiding under utilized cache lines that expend more energy than it should.

We plan to enhance our approach to extend the methodology to more aspects of memory performance not only for the scientific application domain but also in other application domains and to other benchmark suites. We also plan to do a verification study using performance counters (*e.g.* PAPI) on real machines not just use simulation.

References

[1] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving Performance via Mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[3] S. Ramaswamy and S. Yalamanchili, "An utilization driven framework for energy efficient caches," in *Proceedings of the 15th International Conference on High Performance Computing*, ser. HiPC'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 583–594. [Online]. Available: http://dl.acm.org/citation.cfm?id=1791889.1791948

[4] S. Kumar, H. Zhao, A. Shriraman, E. Matthews, S. Dwarkadas, and L. Shannon, "Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy." in *MICRO*. IEEE Computer Society, 2012, pp. 376–388. [Online]. Available: http://dblp.uni-trier.de/db/conf/micro/micro2012.html#KumarZSMDS12

[5] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ser. ISCA '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 357–368. [Online]. Available: http://dx.doi.org/10.1145/279358.279404

[6] J. R. Srinivasan, "Improving cache utilisation," 2011.

[7] S. Bird, A. Phansalkar, L. K. John, A. Mercas, and R. Idukuru, "Performance characterization of SPEC CPU benchmarks on Intel's Core microarchitecture based processor," in *SPEC Benchmark Workshop*, Jan. 2007.

[8] T. M. Conte and W.-m. W. Hwu, "Benchmark characterization," *Computer*, vol. 24, no. 1, pp. 48–56, Jan. 1991. [Online]. Available: http://dx.doi.org/10.1109/2.67193

[9] S. Li, L. Qiao, Z. Tang, B. Cheng, and X. Gao, "Performance characterization of spec cpu2006 benchmarks on intel and amd platform," in *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*, vol. 2, March 2009, pp. 116–121.

[10] J. Poovey, T. Conte, M. Levy, and S. Gal-On, "A benchmark characterization of the eembc benchmark suite," *Micro, IEEE*, vol. 29, no. 5, pp. 18–29, Sept 2009.

[11] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the spec cpu2006 benchmark suite," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 412–423, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1273440.1250713

[12] ——, "Subsetting the spec cpu2006 benchmark suite," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 69–76, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1241601.1241616

[13] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the amd opteron processor," *Micro, IEEE*, vol. 30, no. 2, pp. 16–29, March 2010.

[14] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 209–220, May 2002. [Online]. Available: http://doi.acm.org/10.1145/545214.545239

[15] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting cache line size to application behavior," in *Proceedings of the 13th International Conference on Supercomputing*, ser. ICS '99. New York, NY, USA: ACM, 1999, pp. 145–154. [Online]. Available: http://doi.acm.org/10.1145/305138.305188

[16] W. Alkohlani, J. Cook, and N. Siddique, "Insight into application performance using application-dependent characteristics," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, ser. Lecture Notes in Computer Science, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Springer International Publishing, 2015, vol. 8966, pp. 107–128. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17248-4_6

[17] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[18] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1964218.1964225

[19] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 190–200. [Online]. Available: http://doi.acm.org/10.1145/1065010.1065034

[20] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: Design alternative for cache on-chip memory in embedded systems," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, ser. CODES '02. New York, NY, USA: ACM, 2002, pp. 73–78. [Online]. Available: http://doi.acm.org/10.1145/774789.774805