Efficient HPC Data Motion via Scratchpad Memory

Kayla O Seager, Ananta Tiwari, Michael A. Laurenzano, Joshua Peraza, Pietro Cicotti, Laura Carrington

Performance Modeling and Characterization Laboratory (PMaC)
San Diego Supercomputer Center
University of California, San Diego
kseager@ucsd.edu, {tiwari,michaell}@sdsc.edu, jperaza@cse.ucsd.edu, {pcicotti,lcarring}@sdsc.edu

***Abstract*— The energy required to move data accounts for a significant portion of the energy consumption of a modern supercomputer. To make systems of today more energy efficient and to bring exascale computing closer to the realm of possibilities, data motion must be made more energy efficient. Because the motion of each bit throughout the memory hierarchy has a large energy and performance cost, energy efficiency will improve if we can ensure that only the bits absolutely necessary for the computation are moved through the hierarchy. Toward reaching that end, in this work we explore the possible benefits of using a software-managed scratchpad memory for HPC applications. Our goal is to observe how data movement (and the associated energy costs) changes when we utilize software-managed scratchpad memory (SPM) instead of the traditional hardware-managed caches. Using an approximate but plausible model for the behavior of SPM, we show via memory simulation tools that HPC applications can benefit from hardware containing both scratchpad and traditional cache memory in order to move an average of 39% fewer bits to and from main memory, with a maximum improvement of 69%.**

I. Introduction

DARPA commissioned report recommends the power wall for exascale systems to be at 20 MW [1]; this power wall is already being approached by current petaflops systems (e.g. Japanese K computer's power requirement is at 12.6 MW [2] for 10.5 Linpack petaflops). The 20 MW power envelope for exascale systems translates to a goal of 50 gigaflops/watt in efficiency, a jump of at least two orders of magnitude from the last generation of supercomputers which had an efficiency of about 0.25 gigaflops/watt [3].

One area to reduce some of the energy consumption is by focusing on one of the most energy-demanding aspects of modern day hardware – data movement between the CPU and memory subsystems. The next generation exascale systems will have significantly more compute cores per node than present day systems. This increase in the number of cores will likely result in reduced shared on-chip cache budgets, thereby increasing the frequency of data evictions from those caches. As a result, data movement energy costs will continue to claim a larger share of the overall energy costs required to run a supercomputer with current cache-based systems. The situation will only be exacerbated by a general move of HPC towards more data intensive applications.

Cache-based systems can also waste energy because they are managed entirely by hardware. The design of hardware assumes the principles of spatial locality[1] and temporal locality when moving memory. Unfortunately, these assumptions do not always hold; it is an ongoing problem to attempt to transform algorithms, implementations, compilers and optimization tools to strengthen them, however the effort required and sometimes the very nature of many scientific problems can prevent the development cache friendly applications. If a program has poor spatial locality, the cache will move more data than it actually uses thereby incurring unnecessary costs in terms of both performance and energy.

Software managed on-chip scratchpad memory (SPM) has the potential to mitigate the problem of expending energy moving unused bits. SPM allows programmers to optimize data motion by allowing them to exercise full control over what bits to move and when to move them. However, such management also transfers the difficult task of controlling data movement onto the software. If the programmer must account for this in writing a program, this can pose a significant productivity challenge for the programmers of HPC applications and for the hardware designers looking to investigate the potential benefits of SPM for HPC applications. Our research seeks to help address parts of this challenge by answering the following fundamental questions via simulation – *Do HPC workloads benefit from SPM? If so, are there compute patterns which get the most benefit?* Benefit in this context is quantified in terms of the raw amount of data movement savings when one uses SPM vs. traditional caches. This paper presents a preliminary evaluation of our tools to understand the types of investigations that are possible with the framework that we are developing.

This work also looks at the possible benefit of a hybrid system which has both hardware- and software-managed local memories. Our results show that such a hybrid system can result in decreases of as much as 69% of the data movement of a hardware-managed cache. Finally, we present some preliminary work on the characterization of compute patterns or *idioms* and using those characterizations to guide the mapping of sections of large-scale application onto either of the available local memories available in a hybrid system.

[1]The fact that majority of the widely used chipsets from both the Intel (e.g. Nehalem, Xeon, Sandybridge) and AMD (e.g. Shanghai, Istanbul, Interlagos, Magny Cours) all use 8-word cachelines furthers this claim of how the principal of spatial locality is very much built into the design of modern day caches.

II. Related Work

Researchers in embedded systems have extensively investigated the energy and space benefits of using SPM. Their research has not only examined the design space parameters (e.g., optimal SPM size for different types of embedded applications) but has also looked into developing compiler and runtime solutions (e.g. data tiling, dynamic allocation of most used data to SPM) to optimize the usage of SPM [4], [5], [6], [7]. While HPC workloads differ significantly from workloads that run on embedded systems, we are encouraged by the observation that the compiler community is actively working with the system designers and embedded application programmers to optimize and automate the usage of SPM.

The HPC community's interest in SPM is rather new and is mostly driven by SPM's memory predictability and power savings. Optimal use of a SPM is still an ongoing research. Work presented in [8] profiles sparse matrix multiplication application for data structures that tend to do poorly on cache. Those hot-spots are ported to use SPM with an offline staging algorithm. Our study attempts to paint a general picture of the utility of SPM for HPC workloads rather than focusing on a specific problem. We do so by studying the interactions between key HPC computational patterns and SPM.

SPM has also made its entrance within the HPC realm in the design of heterogeneous/accelerator architectures. IBMs CELL processor uses scratchpads for its local storage and researchers have profiled it for HPC computations and have proposed optimization strategies [9]. Kondo et al. propose SCIMA, which is a VLSI architecture that integrates software managed scratchpad memory on-chip, and analyze its performance on HPC applications whose data set is too large to fit into on chip memory [10]. Finally, GPU's also utilize software managed scratchpad and efforts have been invested in developing and optimizing data management techniques to make automate allocations to SPM and to better use of SPM [11].

Before answering the question of how to better make use of SPM or before designing specialized HPC architectures with SPM, there is the basic question of to what extent HPC workloads really benefit from SPM based systems. In this work, we answer that question and we do so without limiting ourselves on any particular application or architecture domain.

III. Simulation Framework and Methodology

This section describes the simulation infrastructure and the methodology used to compute data movement statistics for the two on-chip local stores - cache and SPM.

A. Simulation Infrastructure

The memory simulators used to compute data motion statistics are built on-top of the PEBIL [12] toolkit, an open source binary instrumentation toolkit for x86/Linux. There are two main components of the simulator infrastructure – an execution analysis tool and a memory simulation tool. The execution analysis tool sets the stage for memory simulation by providing static and dynamic information about the control flow units of the application including instruction types and counts, information on loop and function memberships, relationships to other basic blocks, memory access sizes, and basic block visit counts. Based on the basic block visit counts, the set of the most dominant basic blocks are selected as candidates for memory simulation. The memory simulation tool then instruments the application binary to send the memory addresses from the dominant basic blocks to a series of cache simulation and memory analysis tools. In particular, these address streams are simulated through traditional cache, scratchpad memory, and cache/scratchpad hybrid memory structures that are described in greater detail shortly.

In order to perform these simulations on large scale production runs of HPC applications, the memory address streams are simulated on the fly. This keeps the time and space complexity of dealing with the address streams in check. The simulation tool works on the application's binary, enabling the simulation of multiple memory configurations without touching the application's source code or any other intermediate form of the application. Therefore this enables cache and SPM simulations on real production runs of applications. The overhead of the simulator is also small relative to similarly functioning tools.

B. Methodology

In order to calculate the data movement associated with SPM, we define a memory structure which is a fully set associative cache of a given size with a line size of one word (8 bytes) and uses the least recently used (LRU) replacement policy. This configuration is compared to a traditional cache that is defined as an 8-way set associative cache with 8 word (64 byte) lines and uses the LRU policy. Compared to a traditional cache, a scratchpad of this form lacks the capacity to benefit from spatial locality because it loads only individual words as they are used.

There are a few noteworthy differences between the simulated scratchpad described above and an actual scratchpad. In an actual scratchpad, explicit directives are likely to be required to move important data in bulk prior to the execution of the section of code that utilizes that data. That data is then kept in the local memory until all computations on that data completes. An important consequence of this usage model is that the size of the program's working set must be known in order to take full advantage of the real scratchpad. In our simulation that utilizes a fully associative 1-word line cache to mimic the behavior of scratchpad, data is moved at one word granularity and only when that word is referenced by the program. If the working set size of the computation is smaller than the scratchpad structure, our simulated scratchpad will move the same amount of data as a real scratchpad. However, if the working set does not fit in the scratchpad our simulated scratchpad can *overestimate* the amount of data moved. This over-estimation is the result of the cache line replacement policy (LRU) chosen for our simulation. LRU can prematurely evict words from the simulated scratchpad, only to reload them at a later time when those words are touched again by the computation. In a real scratchpad we note that careful memory and data partitioning (or tiling) and other scratchpad-friendly

loop transformation techniques would be employed by the compiler [13] or the programmer to minimize this behavior, though fully understanding how to best to accomplish this remains an open question.

C. Quantifying Data Motion

In this work, we examine a single-level on-chip cache, scratchpad and hybrid memory structures and estimate the data movement to those structures by counting the number of misses. Data movement for a particular structure is therefore the product of the size of a line and the number of lines that are moved, which by definition is the number of cache misses. To capture the data movement of an HPC application, we instrument basic blocks accounting for a minimum of 95% of the dynamic memory activity in the application and simulate the memory behavior of those blocks through cache, scratchpad and hybrid memory structures. We describe the results of these quantifications of data motion for a series of HPC benchmarks and applications in the next section.

D. Hybrid Cache/Scratchpad

In order to simulate the behavior of hardware comprised of a hybrid of traditional hardware-managed cache and software-managed cache, we first decide how to apportion memory work to each of those structures. Here we use a simple scheme which partitions the memory work by basic block and attempts to use the structure (SPM or cache) per basic block which minimizes data movement. During simulation the memory accesses made by a particular set of basic blocks are handled by a scratchpad structure while another set of blocks is handled by a traditional cache structure.

IV. Scratchpad Memory and HPC Applications

We first consider the extent to which software-managed scratchpad cache can actually reduce the data movement requirements of HPC applications. We expect the data movement benefits of SPM to be most pronounced in cases where hardware-managed traditional cache unnecessarily moves a line into memory using the principle that the remainder of that line is likely to be used before it is evicted. That is, this question reduces principally to one of recognizing how well the application in question takes advantage of spatial locality. To answer this question, we utilize the simulation tools discussed in Section III-B to understand the effects of running on either a scratchpad memory device or a traditional hardware-managed cache.

The HPC applications utilized in this study along with brief descriptions are given in Table I. These applications include two implementations of the sequential Graph500 benchmark, HYCOM, SMG2000 and three Sequoia Benchmark codes — SPHOT, UMT and AMG2006. These applications are designed to run on large-scale systems and the analysis of their data motion behavior on traditional cache and software managed scratchpad should be of great interest to both the application scientists and the hardware designers.

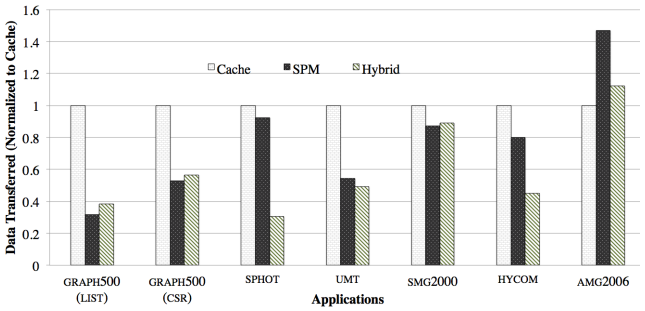

Fig. 1: Total data transferred during an application execution for hardware-managed cache, software-managed SPM and cache-scratchpad hybrid memory structures, normalized to the data transferred for cache.

Figure 1 shows the data movement totals of these applications for three different memory configurations – a traditional cache, a scratchpad memory structure and a hybrid configuration. Detailed descriptions of these three configurations are given in Table II. Per application in Figure 1, the data movement totals are normalized with respect to the total data transferred using cache. Therefore a value of less than 1 for scratchpad memory means that scratchpad moves less data than cache. Ideally, scratchpad-based systems should never move more data than the cache-based systems. However, as we discussed in Section III-B, our simulation setup for scratchpad can overestimate data motion where local memory thrashing occurs due to the working set size being larger than the size of the scratchpad. For a line to be precluded from eviction from cache via LRU policy, referencing only one out of the eight words in the line is enough. For scratchpad, however, since it is one word line, LRU can prematurely evict words only to move them again later when they are referenced again. We observe this behavior in AMG2006. However, apart from this case all other applications benefit significantly from scratchpad – with an average data movement savings of 22%.

Next we turn our attention to the hybrid structure, one in which each basic block is able to make use of the structure which minimizes the amount of data that must be moved on its behalf. As can be seen from Figure 1, the hybrid system which has half of its memory capacity dedicated to scratchpad memory shows dramatic reductions in the amount of data movement required for the entire application. On average, data movement is reduced by 37%, with a maximum of 69% reduction in data transferred for sphot. We attribute this savings to the lack of spatial locality in the memory references issued by the basic blocks that were mapped to SPM in a hybrid system. Extending the memory simulation tool to quantify the spatial locality is an ongoing work. We note that the locality scores can point towards how to better take advantage of the hybrid system's SPM. Based on these scores, we can rank order application's basic blocks that benefit the most from SPM and port only those to use SPM.

These experimental results suggest that there are substantial portions of HPC applications that can save unnecessary data movement by utilizing scratchpad memory despite the fact

TABLE I: Names and descriptions of subject applications used for cache and scratchpad experiments.

Application	Description
Graph500 [14]	Code to construct and traverse weighted undirected graph
HYCOM [15]	Hybrid isopycnal-sigma-pressure (generalized) coordinate ocean model code.
SMG2000 [16]	Parallel semicoarsening multigrid solver.
SPHOT [17]	(Sequoia Tier 3) Monte Carlo scalar photon transport code
UMT [17]	(Sequoia Tier 3) Unstructured-mesh deterministic radiation transport code.
AMG2006 [17]	(Sequoia Tier 1) Algebraic mult-grid linear system solver for unstructured mesh.

TABLE II: Descriptions of the memory configurations used in data motion experiments.

Title	Cache Size (KB)	Cache Assoc.	Cache Line Size (Bytes)	SPM Size (KB)	SPM Assoc.	SPM Line Size (Bytes)
Cache	64	8	64	-	-	-
Scratchpad	-	-	-	64	full	8
Hybrid	32	8	64	32	full	8

that our scratchpad simulation model can overestimate data movement. Moreover, the fact that the data motion reductions resulting from a scratchpad/cache hybrid are so dramatic implies that more work is needed in order to fully understand the types of compute and memory access patterns that benefit from scratchpad memory. In the next section, we present some preliminary work which demonstrates a possible methodology for examining this issue.

V. Idiom Characterization: Early Results

HPC applications can be large and complex and developing a method to break this complexity into smaller and more manageable pieces can aid in characterizing the application's data movement behavior. One such method is to break an application down into its constituent idioms, where an *idiom* is defined as a local pattern of computation or memory access. Examples of idioms are matrix-matrix multiplication, matrix transposition, an array gather/scatter, etc. Understanding where these idioms are present in an application can help us understand how the application is likely to stress the memory subsystem while executing those idioms. In particular, predictions could be made about how different types of memory structures (cache, SPM or hybrid) are likely to interact with a given idiom, which can be used as the basis for making judgments about which of those structures best suits the idiom.

We provide the pseudocode for the gather and scatter idioms in Figure 2. The gather idiom "gathers" data from a potentially random series of memory addresses over a particular range. In the pseudocode, the random accesses are created using an index array `C`. The scatter idiom can be thought of as the converse of gather, meaning results are sequentially scattered from an array into potentially random memory locations covering a particular range. In this work we focus exclusively on the gather and scatter idioms due to their ability to stress memory performance (caused by randomness and indirection in the address stream) and their prevalence in HPC applications.

```
for i= 1 to N, stride
   A[i] = B[C[i]]
```
(a) Gather

```
for i= 1 to N, stride
   A[C[i]] = B[i]
```
(b) Scatter

Fig. 2: Psedocode for Gather/Scatter Idioms

Characterizing the idioms involves studying different ways these idioms can interact with the system components. For gather/scatter, we select a working set size (or memory footprint) that exceeds the size of the available on-chip memory

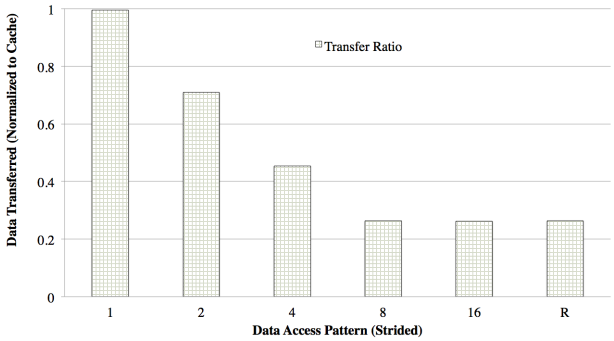

Fig. 3: Amount of data transferred to scratchpad during the gather idiom as a fraction of the amount data transferred to cache during the same code. X-axis shows different values for `stride` variable in Figure 2.

and vary the stride pattern (see `stride` loop stepping variable in Figure 2). For each stride pattern, we simulate the address stream through the cache and scratchpad simulation tools. Figure 3 presents the amount of data transferred to scratchpad during the gather idiom as a fraction of the amount data transferred to cache for the same code. Recall that a smaller ratio than one favors scratchpad because it moved less data than cache. We vary the stride pattern in the x-axis ("R" is random access pattern). The simulations produced the expected results with a significant reduction in data motion for the scratchpad with large strides and random access patterns. The lowered data movement translates directly in energy saving. Despite the strided or random access, with a cache the data transfer is always done at line granularity, whereas with the scratch pad only the relevant data is transferred from memory. The figure shows how data transfers increase for the cache as the stride increases, until only one word out of eight per line is utilized. The results for the same experiment using scatter idiom are almost identical.

We now explain how idiom characterization can fit into the broader goal of optimizing data movement for large scale applications on a cache/SPM hybrid system. To do that, we utilize the GCC plugin PIR [18], [19]. PIR is a source code analysis tool that automatically identifies user-defined compute and memory access patterns within application source code. We used PIR to identify the gather and scatter idioms within HYCOM, an ocean modeling code. PIR identified 33 instances of gather and scatter in the source code for HYCOM. We

then isolated the basic blocks in the HYCOM binary associated with these recognitions, used the memory simulation tools to examine their memory behavior and data movement characteristics. The results of this study show that by using a scratchpad memory instead of cache for all instances of gather/scatter within HYCOM we could reduce the amount of data movement by 20%. When we simulate HYCOM's gather/scatter idioms on traditional cache configuration, total data moved is 94 MB as opposed to 74.5 MB when those idioms' data motion is simulated on a scratchpad.

VI. Discussion and Ongoing Work

Our goal in this paper was to investigate if HPC workloads benefit from the availability of SPM and if so, quantify the data movement savings. The results show that large scale HPC applications can reduce the amount of data moved substantially with the use of scratchpad as the on-chip memory. While there are still many research challenges that need to be addressed before a hybrid system becomes a reality, we believe that the results that we have presented in this paper should spark a dialogue between the hardware and software communities within HPC towards tackling the challenges.

The preliminary results that we have presented in this paper give us multiple avenues to extend this work. Our ultimate goal is to provide a simulation framework that can automatically isolate data motion sensitive idioms from large scale applications and use the idiom characterization to identify what sections of code run well on cache and scratchpad. Below, we itemize some of the ongoing work.

- Extending the simulation framework to more closely mimic the behavior of the real scratchpad and remedy its data movement over-estimation problems for scratchpad.
- Adding locality quantification ability to the current framework to provide porting hints to application programmers in terms of which code-sections to port to SPM. This reduces the amount of labor required to port the code to use software managed caches. Note that porting hints can also be provided utilizing the idiom characterization data.
- Extending the framework to automate the breaking down of large scale applications to their constituent idioms and then using the idiom characterization data as the basis for making judgments about which on-chip memory best suits the idioms.

VII. Conclusion

The path to exascale is laden with hard challenges. Of these challenges energy efficiency is perhaps the most difficult challenge. In this work, we studied the viability of software managed on-chip memory for HPC workloads. We showed the evidence that having both software managed and the traditional hardware managed caches can benefit HPC applications in optimizing and reducing the overall the data motion. Reduction of data movement should translate directly to saving energy.

Our results show that scratchpad can be beneficial to data intensive HPC applications. Running a set of HYCOM idioms that exhibit highly indirect memory access patterns on scratchpad reduced the data movement for those patterns by 20%. Hybrid systems also showed promise for data intensive HPC workloads and, on average, saved 39% in data movement.

Acknowledgment

This work was supported in part by the DOE Office of Science through the Advanced Scientific Computing Research (ASCR) award titled "Thrifty: An Exascale Architecture for Energy-Proportional Computing".

References

[1] P. Kogge, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," *CSE Dept. Tech. Report TR-2008-13*, 2008.

[2] "TOP500 List," http://www.top500.org, June 2012 Report.

[3] R. Gioiosa, "Towards sustainable exascale computing," in *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, sept. 2010.

[4] 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.

[5] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, 2002.

[6] N. Deng, W. Ji, J. Li, F. Shi, and Y. Wang, "A novel adaptive scratchpad memory management strategy," in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE International Conference on*, aug. 2009.

[7] M. Kandemir, J. Ramanujam, M. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, "Dynamic management of scratch-pad memory space," in *Design Automation Conference, 2001. Proceedings*, 2001.

[8] A. Yanamandra, B. Cover, P. Raghavan, M. Irwin, and M. Kandemir, "Evaluating the role of scratchpad memories in chip multiprocessors for sparse matrix computations," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, april 2008.

[9] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, "The potential of the cell processor for scientific computing," in *Proceedings of the 3rd conference on Computing frontiers*, ser. CF '06. New York, NY, USA: ACM, 2006.

[10] H. Okawara, "Scima: Software controlled integrated memory architecture for high performance computing," in *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, ser. ICCD '00, Washington, DC, USA, 2000.

[11] M. Moazeni, A. Bui, and M. Sarrafzadeh, "A memory optimization technique for software-managed scratchpad memory in gpus," in *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*, july 2009.

[12] M. Laurenzano, M. Tikir, L. Carrington, and A. Snavely, "Pebil: Efficient static binary instrumentation for linux," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, march 2010.

[13] M. Kandemir, I. Kadayif, A. Choudhary, J. Ramanujam, and I. Kolcu, "Compiler-directed scratch pad memory optimization for embedded multiprocessors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 3, march 2004.

[14] "Graph 500," http://www.graph500.org/.

[15] "HYbrid Coordinate Ocean Model," http://www.hycom.org.

[16] P. N. Brown, R. D. Falgout, and J. E. Jones, "Semicoarsening Multigrid on Distributed Memory Machines," *SIAM J. Sci. Comput.*, vol. 21, no. 5, 2000.

[17] "Sequoia Benchmarks," https://asc.llnl.gov/sequoia/benchmarks/.

[18] L. Carrington, M. M. Tikir, C. Olschanowsky, M. Laurenzano, J. Peraza, A. Snavely, and S. Poole, "An idiom-finding tool for increasing productivity of accelerators," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011.

[19] M. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole, "Modeling and predicting application performance on hardware accelerators," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, nov. 2011.