Mptrace: Characterizing Physical Memory Usage for Chip Multiprocessors

Francesc Guim[†*], Miguel Ferrer[*], Josep Jorba[*], Ivan Rodero[‡*] and Manish Parashar[‡]
[†]*Intel Barcelona, Spain*
[*]*Open University of Catalunya, Spain*
[‡]*NSF Center for Autonomic Computing, Rutgers Univeristy, United States*
Email: {fguim, mferrer, jjorbae}@uoc.edu, {irodero, parashar}@cac.rutgers.edu

***Abstract*—The performance of high performance computing applications depends highly on how they are implemented. However, their runtime behavior is tightly coupled with the resources they are allocated such as in which cores the application threads run or in which memory devices their memory space is placed. Thus, depending on their characteristics, applications may exhibit more affinity to specific types of processors or perform better in a given architecture. In this paper, mptrace, a novel PIN-based tool aiming at profiling the applications physical memory usage is presented. Mptrace provides mechanisms to characterize the applications access to physical memory pages and thus to explore possible memory usage optimizations such as those aiming at improving performance-power trade offs. Two different implementations of mptrace are described and evaluated using NAS parallel benchmarks. The study of physical memory usage of NAS parallel benchmarks along with the discussion of a specific use case shows the large potential of mptrace.**

***Keywords*-PIN tool; memory profiling ; pagemap; NAS Parallel Benchmarks; High Performance Computing.**

I. INTRODUCTION

High Performance Computing (HPC) evolved over the past decades into increasingly complex and powerful systems. Reaching exaflops computing performance by the end of the decade require the development and deployment of complex and massive parallel processors with multiple cores (e.g., chip multiprocessors) and/or heterogeneous units [1] (e.g., the IBM/Sony Cell processor). The rapid increase in the number of cores has been accompanied by a proportional increase in the DRAM capacity and bandwidth, which presents many challenges such as performance-power trade offs and new programming challenges.

In order to achieve sustained performance and fully tap into the potential of these architectures, the step that maps computations to the different elements must be as automated as possible. In a coarse grain, applications can be classified as memory or processor bound. While the first type of applications is memory bandwidth greedy applications, the second one is mainly limited by either the processor parallelism level or by the amount of computational power that they require. In both cases, mapping computations to appropriate elements (e.g., physical memory) is an important task for two main reasons: (1) ensuring application's performance is crucial from the user perspective, and (2) maximizing the system utilization may improve the system throughput.

When applications use only a subset of all the resources available they waste a substantial amount of power and prevent other applications from taking advantage of the resources, and the system can perform different actions such as allocating the applications in the resources that best match their requirements or reducing the amount of power provided by the resources (e.g., using dynamic voltage and frequency scaling). However, the implementation of such techniques requires profiling methods that are fundamental to understand the applications behavior. Different existing tools provide mechanisms to instrument and gather runtime information. Among them, PAPI [2] provides an interface for collecting low level performance metrics (e.g., number of L2 misses) from hardware performance counters. Other tools are such as Intel PIN [3] provide information related to the application performance. They are able to intercept the application execution flow and to provide information regarding the application performance. Both type tools can be used to characterize the application in terms of processor performance (instructions executed), cache and memory performance (L1 hit rate, L2 hit rate, L2 misses per kilo instructions, etc.) and network usage (link utilization, etc.). Some of them also provide information regarding the virtual addresses used by applications, however, none of these tools provide ways to characterize accurately the physical memory usage and thus how the different channels to physical memory are used.

In this paper we present mptrace, a PIN-based tool, which is able to provide the physical pages that are used by processes on run time. It can be also used to extrapolate other meaningful information such as the usage of the different channels to physical memory. This can help to design new architectures and techniques to optimize the memory usage, thereby improving important aspects such as performance-power trade offs. The main contributions of this paper are: (1) the design and implementation of the mptrace tool, which extracts the mapping between the virtual memory address thread space and the physical memory space, and (2) the study of physical memory usage of NAS Parallel Benchmarks (NPB), which shows the large potential of mptrace.

The rest of the paper is organized as follows: in Section

 ISBN: 978-1-61208-161-8

II the background and related work are discussed; in Section III the mptrace tool is described in detail; in Section IV the evaluation using the NPB is provided as a use case; finally, conclusions and future work are discussed in Section V.

II. Background and Related Work

Previous approaches tackled the characterization of applications mainly from a performance perspective. Existing tools such as PAPI [2], Vampir [4], Paraver [5], Intel vTune [6] and PPW [7] allow to instrument applications by gathering runtime information for both application and computing resources. Existing approaches characterized how applications perform on top of specific hardware. Tools like PAPI, Vampir or Paraver allow instrumenting applications and gathering hardware counters for their executions and extracting information about how they behave. These tools are especially interesting to detect regions of the application that can be improved or to detect system bottlenecks. Other tools do not require instrumenting the application. For example PIN-based tools [3] are able to run non-instrumented binaries and intercept all the stream of instructions prior their execution. These tools are able to gather information concerning the instruction that is about to be executed (i.e., instruction type, operands, etc.).

Over the last years processors have evolved to become very energy efficient supporting multiple operating modes and thus power management techniques have become subject of study. At a very coarse level, power management at server systems level has been based on monitoring load and shutting down unused clusters or transitioning unused nodes to low power modes [8]. Dynamically varying the voltage and frequency proportional to system load has also proved to be effective in reducing energy consumption [9][10]. Dynamic Voltage and Frequency Scaling (DVFS) provides power savings at the cost of increased execution time. Other approaches conducted the power management techniques at the processor level. For example, Cai et al. [11] propose a DVFS techniques based on the hardware thread runtime characterization. These approaches have been developed on top of tools that allowed them to dynamically gather information about the system and the applications. However, the data used to apply DVFS techniques is only from the processor, network and cluster.

The previous approaches tackled the energy consumption optimizations focused on the computing elements. However, memory devices have begun to significantly contribute to overall system energy consumption, and like processors, DRAM devices currently have several low power modes. Delaluz et al. presented software and hardware assisted methods for memory power management. They studied compiler-directed techniques [12], as well as OS-based approaches [13] to determine idle periods for transitioning devices to low-power modes. However, this is not going to be effective in multi-core systems. Cho et al. [14] studied assigning CPU frequencies for DVFS that are memory-aware because the focus of all prior work was on optimal assignment of frequencies to CPU ignoring memory.

Existing tools have already provided mechanisms to understand how applications use the main memory. Some of the previously discussed tools provide information about the hit rate that applications have in L2. This information can be combined with other metrics, for instance the cycles per instruction to estimate the bandwidth that the application requires to the memory (for instance using the misses per kilo instructions). Other trace-based tools can be used to get similar information. For example the PIN-based tool CMPSim [15][16]. It is a PIN [3][17] tool that intercepts memory operations that are fed to a chip multiprocessor cache simulator. The model implements a detailed cache hierarchy with DL1/IL1, UL2, UL3 and memory, and can be configured to model complex cache hierarchies (e.g., a SMP machine of 32 cores sharing the L2 and L3). However, all the previous tools cannot provide more detailed information about how the physical memory is used, such as the memory bandwidth requested to specific memory channels. To do this, these tools would need to provide information about which physical memory locations are mapped to the virtual regions for the application process.

III. Mptrace

The Intel PIN [3][17] project aims to provide dynamic instrumentation techniques to gather information about the instructions that applications execute. PIN API provides mechanisms to implement callbacks that are called once specific events occur on the execution of the target application (i.e., execution of memory operation). Thus, a PIN tool can be build on top of this API to collect a subset of all the available information. This tool can be executed with different applications and there is no biding with specific binaries. Thus no instrumentation is required to the target application of study.

As of today, many tools have been build on top of PIN such as CMPSim [15][16], which is a cache hierarchy simulator that intercepts memory accesses and simulates its accesses using a cache model. Other tools that profile the applications memory access can be found in the PIN software development kit. However, no PIN tool or similar instrumentation tool has been provided to profile the physical memory accesses that applications request. Mptrace is a PIN-based tool that allows intercepting the processes memory accesses and translating the virtual addresses to physical addresses.

Mptrace has two different mechanisms to translate the virtual addresses to physical addresses. The first one is based on the *pagemap* file system, which is a relatively recent mechanism in linux kernel. This file system provides information about the physical location for the given virtual address of a process. The second mechanism is based on

 ISBN: 978-1-61208-161-8

a linux kernel module that translates the virtual addresses to physical addresses without requiring an operating system that supports the *pagemap* file system. Specifically, the kernel module uses *ioctl* system calls to obtain the address for a given page, and does the *walkpage* through the linux memory structures to do the translation. In the following subsections the two mechanisms are discussed in more detail along with the description of the information provided by mptrace.

A. The pagemap version

The first mechanism uses the *pagemap* file system to translate the virtual address to physical address. The *pagemap* file system was released in the kernel 2.6.25 and can be accessed through the */proc/pid/pagemap* filesystem. As it is described in the kernel source, this file allows a user space process to find to which physical frame each virtual page is mapped. It contains a 64-bit value for each virtual page, containing the following data (from *fs/proc/task_mmu.c*, above *pagemap_read*):

- Bits 0-54 page frame number (PFN) if present
- Bits 0-4 swap type if swapped
- Bits 5-54 swap offset if swapped
- Bits 55-60 page shift (page size = 1 "$\ll$" page shift)
- Bit 61 reserved for future use
- Bit 62 page swapped
- Bit 63 page present

Using the *pagemap* system, the mptrace PIN tool provides several functionalities to characterize how the applications access the physical memory pages. The format and information required is highly customizable, it provides information related to cache access (way and set), and memory accesses (physical page address). It also provides ways to reduce the amount of generated information, such as sampling and trace disabling when the application loads data, or the caches are warming up. The current implementation of mptrace provides mechanisms to characterize the memory accesses on the flight. Thus, this PIN tool can provide summarized information about how an application is using the main memory. For example, it provides page access histograms, or clusters of memory regions accessed during an interval of time.

B. The kernel module-based version

The second mechanism has been designed to allow operating systems that do not support the *pagemap* file system, and to improve the mptrace performance as is shown in Section IV. A new kernel module has been developed to carry out the translation of the virtual addresses to physical addresses. To do this it emulates *pagewalk* and process all the different structures provided by the linux memory management unit (MMU) to do the translation. The translation procedures provided by this module are mapped onto specific *ioctl* address. The mptrace kernel module translates a given virtual address to a physical address following the steps listed below.

- Mptrace contacts to the mptrace kernel module using the *ioctl* system call (*IOCTL_GG*) to get the translation for the virtual address @x.
- The kernel module performs the following actions to process the translation:
  - Given the process identifier provided by the user space it looks for the *mm_struct* which contains the information concerning the memory allocated to it.
  - Using the *pgd_offset* kernel function gets the page global directory for @x.
  - Using the *pmd_offset* kernel function gets the page middle directory for @x.
  - Finally using, the *pte_offset* kernel function gets the page table for the address @x. Using the kernel function *pte_page* gets the struct page associated to this virtual address.
  - In order to get the unsigned integer coding the physical address for the resultant struct page, the module uses the function *page_to_phys*.
  - If no error has occurred in the translation the physical address for @x. For example, in those cases were the physical page for the given virtual address is not present, the corresponding error will be returned to the user space.

C. Information and functionalities

As has been discussed in the previous sections, mptrace intercepts all the memory access that the application performs and generates trace files containing information of these accesses. The most representative output data provided by mptrace is described below.

- Current access with respect to the global execution flow: number of global instruction, number of thread instruction, number of memory access instruction, and timestamp.
- Type of memory access: type of access, the instruction pointer for the given instruction, number of operands, and size of the operation in bytes.
- Physical resources used by this operation. For each virtual address it provides: the physical address, cache line and set used by this operation in L1 and L2, and physical page.

Mptrace provides some functionalities that allow both reducing the amount of data generated and summarizing the application behavior such as the number of accesses to the different physical pages. In order to reduce the intrusiveness of mptrace, the structures that it uses have been implemented in a light way fashion (e.g., using lightweight data structures). Moreover, tests to evaluate the level of intrusiveness of mptrace have been conducted. The main

 ISBN: 978-1-61208-161-8

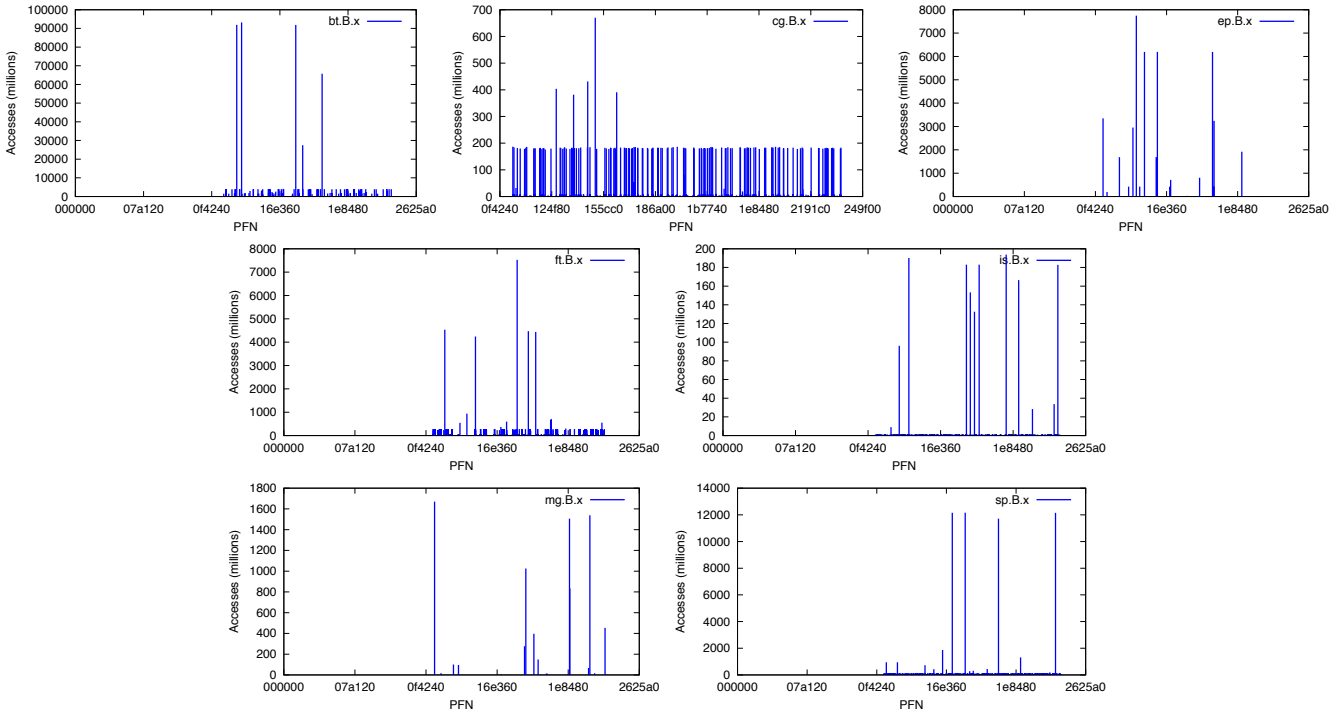

Figure 1: Access pattern of NAS benchmark: BT, CG, EP, FT, IS, MG and SP class B

goal has been to validate that the physical placement for the application virtual space is not modified by the fact that this PIN tool is running. Among other functionalities mptrace allows: sampling, specify in which intervals the memory request have to be processed, which threads have to be instrumented, counting specific events, and dump summarized information (i.e., the number of times that each physical page has been accessed).

IV. Evaluation

In this section, experimental results generated with mptrace are presented. The set of benchmarks that have been used are the NPB that are a set of benchmarks targeting performance evaluation of HPC systems. The goal of this study is to characterize how the different NPB kernels use the physical memory, and to understand how the working sets of these kernels are mapped to the physical pages by the operating system and hardware and how often these pages are accessed. A performance study of the two different implementations presented in this paper is also provided.

A. Methodology

The experiments were conducted with a server with an Intel(R) Core(TM)2 Quad CPU Q9450 processor and 8GB of memory running Linux kernel 2.6.34. The processor provides four hardware threads. NPB were run with the mptrace tool using the *pagemap* file system mechanism. The main parameters considered to conduct the experiments are listed below:

- Each application ran without co-allocation of other applications to avoid interferences with applications requesting memory to the operating system.
- Each application ran with the total amount of hardware threads that the processor provides in order to avoid context switching and other non-desired OS traps.
- Mptrace started tracing at the instruction count 1 million. In these experiments mptrace only accounted for the number of access to the physical pages and thus no other traces were generated (i.e., with the stream of reads and writes to the main memory).

B. Results

Figure 1 presents the number of accesses that each of the physical pages available in the memory device has been accessed by each of the NPB applications. The x-axes show the page number and the y-axes shows the millions of accesses that the application has accessed this page.

The plots show that the amount and distribution of memory access differ for the different NPB kernels. For instance, the MG kernel access few thousands of millions of memory instructions while the BT kernel memory accesses are more than 10 times larger. Since the MG and BT kernels run in 6 and 10 minutes, respectively, the amount of memory accesses per second is substantially different. However, this type of information can be gathered using other traditional tools (i.e., CMPSim). The interesting information that these plots provide is how separated the memory accesses for

 ISBN: 978-1-61208-161-8

each of the NPB applications and the amount of accesses per physical page are from one another. In the case of the CG, the accesses are equally populated among all the different physical pages that are available to the threads. The rest of the benchmarks accesses are located in a relatively small number of physical pages. The MG, FT, EP, and BT benchmarks basically access to few tens of physical pages. However, the amount of accesses is very high for BT (up to 95,000 million access to the same page) with respect to the other two benchmarks (up to 1,800 million accesses to the same page). Therefore, three different type of patterns can be observed in this scenarios: CG does many accesses to many different physical pages; EP, LS and MG do small number of accesses to small subset of pages; and BT does large amount of access to a small subset of pages. Combining this information with time information and cache hierarchy information can lead to interesting characterization of how the memory subsystem is used. Furthermore, as it discussed in the following paragraphs it can derive to some optimizations in the memory system address decoder (i.e., how the virtual memory is placed in the physical memory) and how the memory device is configured (i.e., the amount of frequency that it has to run to deliver the required bandwidth).

As well as defining policies to address important problems such as reducing the memory contention when consolidating workloads, mptrace can be used, for example, to develop novel techniques such as predictive memory power management at run time. We propose using mptrace to extend the work on dynamic memory voltage scaling proposed by Deng et al. [18] considering the ability to select different frequencies for different memory channels as a case of study to show the large potential of mptrace. The process of mapping physical addresses to memory channels to main memory is proprietary to each memory control design. Mptrace can be used to obtain the physical addresses accessed by the applications and then process the data to obtain the channels access patterns. Figure 2 shows the memory access patterns for a large amount of channels (i.e., 64 channels) using two different algorithm for mapping memory physical addresses to channels: (1) *default*, where accesses are clustered to certain channels (e.g., clusters of 256MB), and (2) *interleaving*, where accesses are distributed across different channels. The figures illustrate how the algorithm for mapping physical memory addresses to channels can significantly affect the memory access pattern, and presumable the application behavior. They show that peak memory bandwidth is not always demanded by the application and there is unequal distribution of accesses across channels. This asymmetry and unequal distribution of traffic present opportunities to control the channels independently (i.e., scaling the dynamically the frequency).

In the previous sections two different mechanism to translate virtual addresses to physical address have been

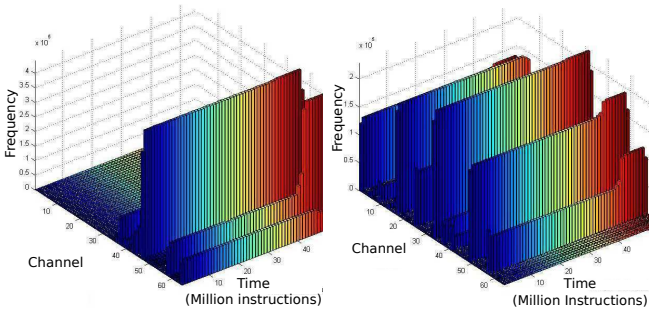

Figure 2: Channels access pattern with default (left) and interleaving (right) mapping policies

introduced: using the *pagemap* file system or using a kernel module. They do not only differ on the resources that they need but also they differ in their performance. As can be observed in Figure 3 the kernel-based implementation is substantially faster than the *pagemap*, especially for short and large runs. This figure presents the number of microseconds that mptrace needed to trace 100K, 1M and 10M of memory instructions for each of the NPB applications. In all of the cases the first implementation performs better than the second one. The difference is especially significant for 10k and 10M memory instructions. Hence the kernel implementation runs two times faster than the other implementation, on average, which is especially important for very large runs.

V. Conclusion and Future Work

In this paper, the mptrace PIN tool, which aims to profile and characterize the physical memory usage for HPC applications, has been presented. Two different implementations of mptrace are described and evaluated using NPB. Specifically, the physical memory usage is characterized by each of the NPB kernels. For each NPB kernel the number of accesses to each physical page is shown. Three different types of patterns are observed in this scenario: (1) CG accesses many times many different physical pages, (2) EP, LS and MG access fewer times a small subset of pages, and (3) BT accesses many times a small subset of pages.

The results show the large potential that mptrace has to study the applications physical memory usage. As of today, many of the tools provide mechanism to understand how the applications virtual space is used; however, information regarding the mapping of virtual addressed to physical memory allows us to understand how the memory devices are used (e.g., to understand the bandwidth required for each of the memory channels). This can lead to designing and optimizing novel architectures and software mechanisms along multiple dimensions such as performance, power and their trade offs.

Current and future research efforts include the development of: (1) a web-based framework to launch, process and generate memory characterization, (2) tools to automatically

 ISBN: 978-1-61208-161-8

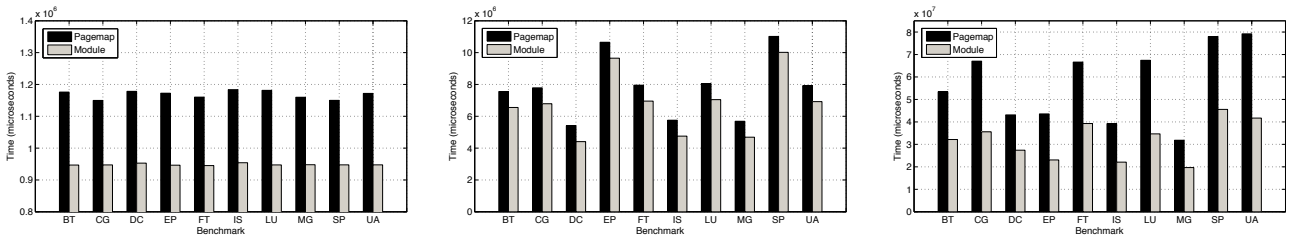

Figure 3: Execution tine of mptrace tracing 100K (left), 1M (middle), and 10M(right) instructions for NPB (OpenMP version)

characterize how the memory is used during the application execution, and (3) techniques to optimize the memory management based on the data provided by mptrace.

ACKNOWLEDGMENT

The authors would like to thank Jose Antonio Martinez, Miquel Perello and Karthik Elangovan for their contributions to this work. This work has been partially supported by the HAROSA Knowledge Community of the Internet Interdisciplinary Institute (http://dpcs.uoc.edu).

REFERENCES

[1] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous Chip Multiprocessors," *Computer*, vol. 38, pp. 32–38, 2005.

[2] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *Int. J. High Perform. Comput. Appl.*, vol. 14, pp. 189–204, 2000.

[3] C. 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," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005.

[4] F. J. Gmbh, I. Bericht, W. E. Nagel, A. Arnold *et al.*, "VAMPIR: Visualization and Analysis of MPI Resources. http://www.pallas.de/pages/vampir.htm."

[5] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "PARAVER: A Tool to Visualize and Analyze Parallel Code," In WoTUG-18, Tech. Rep., 1995.

[6] J. H. Wolf, "Programming Methods for the Pentium III Processor's Streaming SIMD Extensions Using the VTune Performance tool," 1999.

[7] H.-H. Su, M. Billingsley, and A. D. George, "Parallel Performance Wizard: A Performance System for the Analysis of Partitioned Global-Address-Space Applications," *Int. J. High Perform. Comput. Appl.*, vol. 24, pp. 485–510, 2010.

[8] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient Server Clusters," in *2nd international conference on Power-aware computer systems*, 2003, pp. 179–197.

[9] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just in time dynamic voltage scaling: exploiting inter-node slack to save energy in MPI programs," in *ACM/IEEE conference on Supercomputing (SC)*, 2005, p. 33.

[10] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, July 2003.

[11] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González, "Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions," in *International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 240–249.

[12] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "Hardware and Software Techniques for Controlling DRAM Power Modes," *IEEE Trans. Comput.*, vol. 50, no. 11, pp. 1154–1173, 2001.

[13] V. Delaluz, M. Kandemir, and I. Kolcu, "Automatic data migration for reducing energy consumption in multi-bank memory systems," in *39th Design Automation Conference (DAC'02)*, 2002, pp. 213–218.

[14] Y. Cho and N. Chang, "Memory-aware energy-optimal frequency assignment for dynamic supply voltage scaling," in *International Symposium on Low Power Electronics and Design (ISLPED'04)*, 2004, pp. 387–392.

[15] A. Jaleel, R. S. Cohn, C. keung Luk, and B. Jacob, "CMPSim: A Pin-Based On-The-Fly Multi-Core Cache Simulator," in *Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2008.

[16] J. Moses, K. Aisopos, A. Jaleel, R. Iyer, R. Illikkal, D. Newell, and S. Makineni, "CMPSchedSim: Evaluating OS/CMP Interaction on Shared Cache Management," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 113 –122.

[17] K. Hazelwood, G. Lueck, and R. Cohn, "Scalable Support for Multithreaded Applications on Dynamic Binary Instrumentation Systems," in *2009 International Symposium on Memory Management (ISMM)*, Dublin, Ireland, June 2009, pp. 20–29.

[18] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active Low-power Modes for Main Memory," in *6th International conference on Architectural support for programming languages and operating systems*, 2011, pp. 225–238.

 ISBN: 978-1-61208-161-8