2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

Modeling the Energy-Time Performance of MIC Architecture System

Lavanya Ramapantulu*, Thy Dao†, Dumitrel Loghin*, Nam Thoai† and Yong Meng Teo*
*Department of Computer Science
National University of Singapore
13 Computing Drive, Singapore, 117417
Email: {lavanya,dumitrel,teoym}@comp.nus.edu.sg
†Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology
Ho Chi Minh City, Vietnam
Email: {namthoai}@hcmut.edu.vn

***Abstract*—Many Integrated Core (MIC) architecture systems are becoming increasingly popular for HPC applications as they have the dual-advantage of accelerating vector processing and a general-purpose programming model. One of the key challenges for energy-efficient execution on MIC architecture systems is to determine time and energy-efficient configurations among a large system configuration space. Given a parallel program with an execution time deadline and an energy budget, we propose a measurement-based analytical modeling approach to determine these system configurations. In contrast to current approaches, we model both inter- and intra-core resource overlaps, memory contention among threads within a core and memory contention across multiple cores. The model is validated against direct measurement using six representative HPC applications on Intel Xeon Phi system. We show that a Pareto frontier consisting of optimal configurations exist for a parallel program executing on MIC architecture systems. To further understand the Pareto frontier, we use the performance-to-power ratio metric (PPR), that quantifies the amount of useful computations performed per unit energy during the execution. The proposed approach can be used to determine what thread affinity is suitable for offloading execution to accelerators such as Xeon Phi and save energy.**

I. Introduction

With the slowing down of Moore's law and dark silicon limiting the number of active cores in a multi-core processor, mixing CPUs and accelerators is a viable approach to scale-up parallel computing performance. Recent years have seen the wide adoption of accelerators by the HPC community. Among the Top500 systems in November 2015, there were about 104 systems with accelerators [1].

Traditionally, GPUs have been dominating the accelerator arena, but with the launch of Intel's Knight Corner in 2012, we have seen another class of accelerators being adopted mainstream, namely the Many Integrated Core (MIC) architecture. The increasing adoption of this architecture is evident in the Top500 systems, where 32 out of the 104 systems with accelerators use the Intel Xeon Phi coprocessor which is based on the MIC architecture.

With at least 50 cores, Intel Xeon Phi coprocessor adds high parallelism on a single node and has a theoretical peak of two TFLOPS for single precision, one TFLOPS for double precision and over 352 GB/s of memory bandwidth. This performance as well the flexibility to be used both as a coprocessor or a standalone processor, promotes it as a new intra-node heterogeneous platform for HPC applications. In contrast to GPU, Xeon Phi has a general-purpose programming environment and can be programmed with common programming languages, thus making it even more popular among HPC users.

However, from a system perspective, coprocessors based on MIC architecture offer a large system configuration space to execute a parallel application. For a given parallel program, reaching the theoretical peak performance of the Xeon Phi is challenging because it depends on inherent parallelism in the program, how to bind threads to cores, the degree of vectorization and memory usage of the applications. Hence, for a HPC user, determining the optimal system configuration to execute a parallel application is non-trivial and poses a number of research challenges such as:

1) For a given program, what is the number of threads that achieves the best performance?
2) For a given program and number of threads, what thread affinity mode achieves the best performance?

Answers to these questions help both application developers to gain insights on program hot-spots, and system designers to identify capacity bottlenecks, and thus optimize software-hardware co-design to improve system performance. This paper addresses these challenges and proposes an approach to determine a system configuration to execute a parallel program on MIC architecture system in an efficient manner.

Current approaches to analyze performance of parallel programs mainly include instrumentation or application profiling-based measurement techniques which trace the complete execution of the program on a particular hardware system to identify both application and hardware bottlenecks [2], [8], [20], [22]. However, they are generally intrusive and difficult to generalize across programming languages. Another approach to understand application hot-spots on prospective hardware is the use of cycle-accurate micro-architecture level simulators [3], [6], [17]. However, analyzing application executions even with reasonable input sizes is very time-consuming [14].

This paper presents an approach to determine energy-time efficient system configurations for executing a parallel program using a measurement-driven analytical model. The proposed analytical model is formulated using parametric values obtained from baseline executions of the application to measure workload and architectural artefacts. The key

2375-0227/16 $31.00 © 2016 IEEE
DOI 10.1109/MASCOTS.2016.54

novelties of our approach are modeling both inter- and intra-core resource overlaps and resource contention.

Given a parallel program, the proposed approach determines the system configuration in terms of number of cores and number of threads per core. Thus, this paper proposes a systematic method that can be used to determine the thread affinity mode and the number of threads for efficient execution of a HPC application on MIC architecture systems such as Xeon Phi accelerators.

The proposed model is validated against direct measurements on an Intel Xeon Phi card having a 5110P coprocessor with 60 cores operating at 1.053 GHz using a range of NAS NPB benchmarks, including both kernel and applications [4]. Validation results for all possible configurations show that our model accuracy is within reasonable bounds of less than 15%. As an example, we apply our model to determine energy-time efficient configurations for HPC applications.

Our key contributions are:

1) A measurement-driven analytical model to determine time-energy efficient performance of parallel program. In contrast to current approaches, we model both inter- and intra-core resource overlaps, memory contention within and across multiple cores in a MIC architecture.
2) We show that parallel programs executing on MIC architecture systems exhibit Pareto-optimal configurations which execute in the minimum possible time for a given energy budget or consume the minimum energy for a given execution time deadline.
3) We show the importance of performance-to-power ratio (PPR) metric in determining what thread affinity mode is more suitable for offloading on accelerators with MIC architecture.

The rest of the paper is organized as follows. In Section II, we present the related work. Section III discusses our measurement-driven analytical modeling approach and Section IV validates the model. We present the energy efficiency analysis in Section V and summarize in Section VI.

II. Related Work

We broadly divide previous work into two categories: (i) performance modeling of parallel programs, and (ii) performance studies of MIC architecture systems. We compare and contrast these with our proposed approach.

A. Performance modeling

More recently, at algorithm level, asymptotic analysis-based modeling techniques are used to derive trade-offs between computation and communication [9], [10]. However, the approach presented in this paper is at a lower level of abstraction, so that insights into both application and architecture bottlenecks can be inferred. Closer to this paper's proposed approach are our previous works which use non-intrusive inputs, hardware event counters [24], [25], [31]. While these works focus on parallel programs, they do not consider intra-node heterogeneous systems such as coprocessors based on MIC architecture. This paper focuses on modeling the time performance of systems with coprocessors based on MIC architecture and considers both inter- and intra-core overlaps and resource contention.

Guo et al. [14] also analytically model application behaviour on prospective architectures by capturing the dynamic behaviour of an application using the control flow statistics. However, we use a measurement-based analytical model to characterize application execution for different hardware. Other alternative approaches to predict performance include statistical methods that rely on black-box regression to infer dependencies between hardware parameters and application performance [5], [19], [30]. Our approach is not black-boxed and thus enables analytical prediction of the impact of changing different system parameters on program execution performance.

B. MIC performance studies

Fang et al. [11] presented an empirical study on Xeon Phi, stressing its performance limits and relevant performance factors using micro-benchmarks. The system architecture components that were studied in detail include the vector processing cores, the on-chip memory, the off-chip memory, the ring interconnects and the PCI express connection. They also attempted to provide a simplified machine view for performance tuning and application design. Ramachandran et al. [23] examined the performance of NPB OpenMP version on Xeon Phi and identified some performance optimizations. In contrast, Schmidl et al. [27] evaluated the performance of OpenMP applications on Xeon Phi and compared the performance of the coprocessor with a Xeon-based compute node. Vladimirov et al. [33] studied the performance impact of using coprocessors for MPI-based applications. While these works are purely measurement based studies, this paper proposes an analytical model to analyze MIC architecture performance.

Fang et al. [12] used benchmarking to evaluate the performance of various optimization techniques with a focus on guiding kernel design. By using a set of micro-benchmarks, they characterized the three major components of the Phi architecture - cores, memory, and interconnect. They also synthesized a set of four machine-centric optimization guidelines and a simplified machine model for facilitating kernel design and performance tuning on the Xeon Phi. Ramos et al. [26] proposed communication models for cache-coherent MIC architecture and applied these models to optimize algorithms with complex data exchanges. While these works are related to optimizing the kernel or specific algorithms, this paper focuses on modeling the performance of parallel programs on the MIC architecture.

Heinecke et al. [15] implemented the Linpack benchmark on single and multi-node systems based on Xeon Phi coprocessors in both native and hybrid configurations. Their implementation on Knights Corner employs novel dynamic scheduling and achieves close to 80% efficiency. Si et al. [28] developed a

multi-threaded MPI implementation on many-core environments such as Xeon Phi to coordinate the runtime engine of the threads and share idle threads with the application. In contrast, we focus on modeling the impact of thread affinity on energy-time performance to determine efficient system configurations to execute parallel programs.

III. Proposed Approach

In this section, we discuss our proposed analytical model. We first present an overview of our model and its assumptions and some background on the MIC architecture. Next, we derive the model parameters to determine time and energy performance.

A. Overview

Our objective is to determine a energy and time-efficient configuration for executing a given parallel program on a coprocessor based on the MIC architecture. While the MIC architecture offers immense amount of thread-level parallelism (TLP), it is non-trivial for the user to determine the optimal system configuration for execution among the huge configuration space offered by such a system. A system configuration is defined as a tuple consisting of the number of cores and the number of threads per core[1]. Due to the large TLP offered by the MIC architecture, users of such systems are given different options of utilizing the underlying resources using different thread affinity modes. Thus, the configuration tuple determined using our approach represents the number of threads and the thread affinity mode to be used by the user.

The proposed approach determines the system configuration that has the best execution time performance by determining the optimum number of threads per core (τ) and the number of cores (c) to execute the program. Counter-to-intuition, using the maximum number of threads per core, compact thread affinity, does not necessarily translate to the best time performance. As the number of threads within a core increase, they compete with each other for shared resources such as memory. This is modeled in our approach as intra-core contention. Alternatively, keeping intra-core contention to a bare minimum and scheduling only a single thread per core, scatter thread affinity, may not be optimal as the threads across cores also contend for shared-memory, which we define as inter-core contention. The performance impact of choosing either the policy with (i) maximum number of threads per core (compact) or (ii) single thread per core but using more cores (scatter) is non-trivial and thus our paper addresses this challenge by modeling both intra- and inter-core contention for shared memory.

To infer the program's demands on system resources such as CPU and memory, we characterize the workload using baseline executions to derive program and architectural artefacts. These baseline executions are performed using a small program input size. Using these measurements, we derive useful work cycles and the intra- and inter-core memory contention. The effect of memory contention is observed by measuring the number of stall cycles due to cache misses. This approach is outlined in Figure 1 and the notations used in our approach are described in Table 1.

[1]Considering a single Xeon Phi node can have 60 cores with 4 threads executing per core, and two different thread affinity modes, compact and scatter, result in a total configuration space of $60 \times 4 \times 2 = 480 - 3$ (common configurations) = 477 configurations.

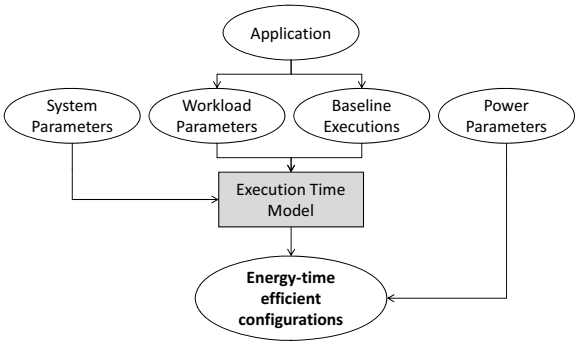

Figure 1: Approach overview

Symbol	Description
Workload Parameters	
$\mathcal{P}$	parallel shared-memory program
$\mathcal{P}_s$	$\mathcal{P}$ with smaller input size
S	scaling factor from program $\mathcal{P}_s$ to $\mathcal{P}$
Baseline Execution	
I_s	instructions in $\mathcal{P}_s$
w_s	work cycles in $\mathcal{P}_s$
b_s	non-memory stall cycles in $\mathcal{P}_s$
$m_{intra,s}$	memory-related stall cycles in $\mathcal{P}_s$
$m_{inter,s}$	memory-related stall cycles in $\mathcal{P}_s$
System Parameters	
c_{max}	maximum no. of cores in MIC architecture system
τ_{max}	maximum no. of threads per core
f	core clock frequency
Time Model	
w	work cycles for $\mathcal{P}$
b	non-memory stall cycles for $\mathcal{P}$
m_{intra}	intra-core memory-related stall cycles for $\mathcal{P}$
m_{inter}	inter-core memory-related stall cycles for $\mathcal{P}$
c	no. of cores per accelerator node executing program $\mathcal{P}$
τ	no. of threads per core executing $\mathcal{P}$
WPI	work cycles per instruction
T_{work}	time for computation
T_{stall}	non-overlapped time due to memory stalls
T_{intra}	non-overlapped time due to intra-core stalls
T_{inter}	non-overlapped time due to inter-core stalls
T	total execution time of a program $\mathcal{P}$

Table 1: Model parameters

B. MIC architecture

An overview of the Many Integrated Core (MIC) architecture is illustrated in Figure 2 using the Intel Xeon Phi coprocessor as an example implementation. The Intel Xeon Phi coprocessor is primarily composed of processing cores, caches, memory controllers, PCI express client logic, and a very high bandwidth, bidirectional ring interconnect. Each core has a 32-KB L1 instruction cache, a 32-KB L1 data cache

and a private 512-KB L2 cache that is kept fully coherent by a globally-distributed tag directory.

The memory controllers and the PCIe client logic provide a direct interface to the GDDR5 memory on the coprocessor and the PCI express bus, respectively. All these components are connected together by the ring interconnect. There are 64 tag directories (TD) connected to the ring. Address mapping to tag directory is based on the results of hash functions of the memory addresses, hence, the memory addresses will be distributed evenly over the ring, which helps better communication. The memory controller (GDDRMC) is distributed symmetrically around the circle. The addresses are distributed evenly across the controllers, thereby eliminating "hot spots" and provide unified access model.

The interconnect is implemented as a bidirectional ring. Each direction consists of three separate rings [13]. The first ring, also the largest and most expensive ring of the three is the data block ring. This data block ring is 64 bytes wide to support high bandwidth. The address ring is smaller and is used to send read/write commands and memory addresses. Finally, the smallest ring and the least expensive ring is the acknowledgment ring, which sends flow control and coherence messages. When the cores access the L2 cache and cannot find the necessary data (cache miss occurs), one request will be sent to the tag directory. If data is located on the L2 cache of another core, a request will be sent to that core and the data will be sent via the data ring. If the requested data is not found in any caches of any cores, a memory address is sent from the tag directory to the memory controller.

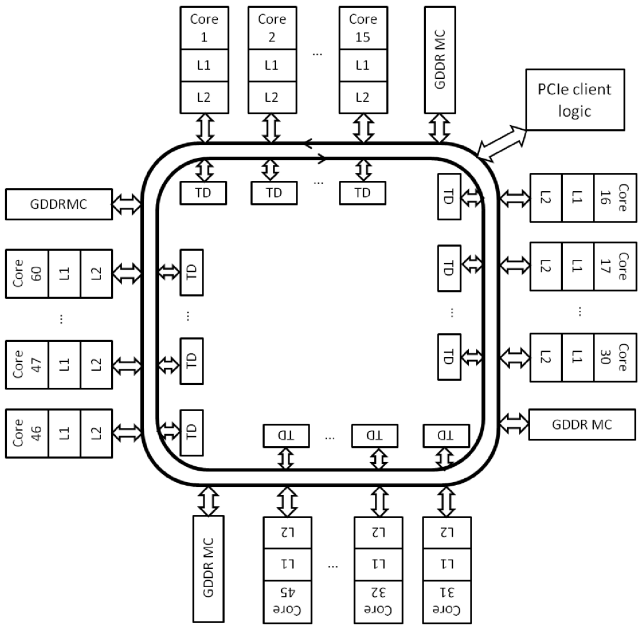

Figure 2: MIC architecture

C. Time performance model

In this section, we derive the execution time, T, for a shared-memory parallel program $\mathcal{P}$, executing on a accelerator based on the MIC architecture. Adding more computational resources to a parallel program splits the computations or useful work among these multiple resources leading to execution speedup but incurs overheads due to thread communication and contention. While intra-core communication using the shared L1 cache is faster, the overheads due to intra-core contention for the L1 cache causes speedup losses in parallelism. Similarly, when only a single thread is scheduled on a single core, it minimizes intra-core contention, but inter-core contention for the L2 memory causes speedup losses. We model both intra and inter-core contention for shared-memory by considering the overlap of these memory accesses with useful work or program computations.

The execution time of a parallel program executing on c cores and τ threads per core depends on the total number of cycles executed by the thread in the critical path of the execution. Thus,

$$T = \max_{\substack{1 \leq \tau \leq \tau_{max} \\ 1 \leq c \leq c_{max}}} T_{\tau,c} \tag{1}$$

The execution time of a thread on a core in MIC architecture systems depends not only on the computation time but also the contention time for shared resources both within and across cores. We model this execution time as service time offered by two system resources in the MIC, namely, cores and memory. However, the response time of these resources overlap and thus the execution time of the thread cannot be determined by a simple summation of the service time and waiting time of both the servers, namely core and memory.

As the MIC cores have deep pipelines to simultaneously compute and access memory, we split the execution time of the thread into the useful work done by the core along with the overlapped memory accesses, and consider the non-overlapped time of core separately. Therefore, we derive this execution time by considering time overlapped with computation (T_{work}) and non-overlapped time (T_{stall}) waiting for a shared resource. Hence,

$$T_{\tau,c} = T_{work} + T_{stall} \tag{2}$$

The execution time on the core and the overlapped memory access time is determined by using the cycles spent in the execution stage of the pipeline, and the core clock frequency. Thus,

$$T_{work} = \frac{cycles_{work}}{f} \tag{3}$$

The number of overlapped cycles between computation and memory access depends on the instruction level parallelism (ILP) of the program and the underlying processing core. We measure this artefact by using a program subset $\mathcal{P}_s$ and determine the useful work cycles executed by a core ($w_s + b_s$). This in turn is determined by measurements using the PAPI [34] hardware counters for cycles and instructions

for the program $\mathcal{P}_s$.

$$WPI = \frac{w_s + b_s}{I_s} \tag{4}$$

We validate in Section IV-B1, our hypothesis that as programs scale from $\mathcal{P}_s$ to $\mathcal{P}$, the Work Cycles per Instruction (WPI) remains approximately the same, as both overlapped work cycles and instructions scale by the same amount for a given program and system, as these are effects of ILP. Thus, the overlapped work cycles for a program $\mathcal{P}$ can be determined using the measured WPI for $\mathcal{P}_s$ and using the scaling factor S for determining the instructions of program $\mathcal{P}$.

$$cycles_{work} = WPI \times S \times I_{\mathcal{P}_s} \tag{5}$$

This scaling of instructions with program input size is validated in Section IV-B2. Next, we discuss the impact of TLP on the execution time, and derive how the model determines the non-overlapped cycles, due to both intra and inter-core contention. We use two separate measured parameters to determine both the contentions. While the first parameter $m_{intra,s}$ measures the intra-core contention for the shared L1 cache using the compact thread affinity mode, the second parameter $m_{inter,s}$ measures the inter-core contention for memory.

1) Intra-core contention: As the number of threads increase within a core, it is expected that the execution time improves due to TLP. But, there are also parallelism losses that happen and for an efficient software-hardware co-design it is imperative to gain insights into these losses. As the number of threads within a core increases, the number of instructions executed per thread decreases due to TLP, but the number of cycles does not decrease by the same ratio due to the waiting time of the memory requests per thread at the L1 cache. Thus, we model the parallelism loss due to contention for the shared L1 cache among the threads within a core by measuring the increase in the number of stall cycles.

We measure the increase in the non-overlapped stall cycles by increasing the number of threads per core for program $\mathcal{P}_s$, and use this to determine the non-overlapped stall cycles $m_{intra,s}$ due to L1 cache contention for a given number of cache accesses. If $\lambda_{intra,\tau}$ is the number of L1 data access requests when τ threads are executing in a single core, the intra-core waiting time and non-overlapped service time (T_{intra}) is

$$T_{intra} = \frac{m_{intra}}{f} = \frac{\lambda_{intra,\tau} \times \alpha_{\tau,s}}{f} \tag{6}$$

where α_τ is the number of stalls per L1 cache access due to τ threads executing within a core for program $\mathcal{P}_s$. We use this time for determining the execution time for a program executing with compact affinity mode when the number of cores is one, and the number of threads vary from one to four. We use the measured values of $\lambda_{intra,\tau,s}$ for the $\mathcal{P}_s$ and derive the parameters for program $\mathcal{P}$ using the scaling factor S as:

$$\lambda_{intra,\tau} = \lambda_{intra,\tau,s} \times S \tag{7}$$

The scaling of the number of data accesses, $\lambda_{intra,\tau}$ is validated in Section IV-B3.

As the program can be compute bound or memory bound in a given core depending on the number of threads being used and the amount of contention, the total execution time of a program $\mathcal{P}$ using τ threads on a single core may be determined as:

$$T_{\tau,1} = T_{work} + T_{intra} \tag{8}$$

While the above determines the execution time for TLP on a single core, it does not consider inter-core contention for shared-memory.

2) Inter-core contention: As the number of threads executing a program increase, based on the thread affinity mode, the number of cores increase and thus they contend for the shared memory. As there are local tag directories (TD) per core, a memory access to the GDDR5 only happens when the requested access misses the distributed L2 cache across all the cores. Thus, the service rate of the inter-core memory requests varies with the number of misses and the number of prefetches. The Xeon Phi uses a hardware prefetching mechanism that dynamically identifies cache miss patterns and generates prefetch requests [18]. Thus the service rate of the memory controller depends on the inter-core memory request arrival rate, which is cumulative from both the prefetcher and misses during execution. As both of these requests queue at the memory controller (GDDRMC), and with λ_{inter} as the total number of memory requests, the inter-core non-overlapped memory response time is:

$$T_{inter} = \frac{m_{inter}}{f} = \frac{\lambda_{inter,c} \times \beta_{c,s}}{f} \tag{9}$$

where β_c is the total number of cache accesses due to c active cores in the MIC architecture system.

We use this time for determining the execution time for a program executing with scatter affinity mode when the number of threads per core is one, and the number of cores vary from one to 60. We use the measured values of $\lambda_{inter,c,s}$ for $\mathcal{P}_s$ and derive the parameters for program $\mathcal{P}$ using the scaling factor S as:

$$\lambda_{inter,c} = \lambda_{inter,c,s} \times S \tag{10}$$

The total execution time of a program $\mathcal{P}$ using a single thread on c cores is determined as:

$$T_{1,c} = T_{work} + T_{inter} \tag{11}$$

The execution time for a program using c active cores with τ threads per core, considering overlap of both intra- and inter-core contention can be derived as the time due to the resource having the maximum response time. Thus, the time due to stalls is determined from both the contentions, and using Equation 2, the execution time is:

$$T_{\tau,c} = T_{work} + max(T_{inter}, T_{intra}) \tag{12}$$

D. Energy Performance

To determine the power consumed by the cores in the MIC architecture, we use PAPI counters and measure the instantaneous power using micpower component. For each of

the programs we measure the micpower:::tot0 [21], a native event of the micpower component and use this value ($P_{\tau,c}$) to compute the energy of the programs executing on the MIC architecture system:

$$E_{\tau,c} = T_{\tau,c} \times P_{\tau,c} \quad (13)$$

IV. Model Parameterization and Validation

The measurement driven inputs to our analytical model are obtained from MIC architecture system characterization and program characterization. We first describe the programs and the systems used for validation of the proposed model. Next, we discuss an extensive validation of the model parameters for each of the hypothesis presented in Section III-C. Next, we present the validation results of the model outputs against direct measurements from hardware counters.

A. Workloads and Setup

While our approach is applicable on generic shared-memory parallel programs, we selected a representative subset of six benchmark programs from NASA Parallel Benchmark (NPB) suite [32] for presentation in this paper. This subset was chosen to represent different program demands on both CPU and memory resources. These programs also exert different inter and intra-core resource demands as shown in the paper. A brief description of the programs and their input sizes for class A is shown in Table 2.

While the table lists the program input size for only class A, the problem sizes for class A,B and C used are in the ratio 1:4:16. We use the OpenMP version of the program and executed it in the native mode on the Xeon Phi coprocessor. The programs are compiled using the Intel compiler for C and Fortran, with full optimizations -O3 and the MIC flag -mmic.

The experiments were conducted on a system consisting of an Intel node with a Xeon Phi card based on the MIC architecture. The host system has 128GB of main memory and a dual-socket Intel Xeon E5-2680 processor operating at 2.7GHz. The Xeon Phi card is based on the 5110P coprocessor with 60 cores operating at 1.053 GHz. The card has 8GB of GDDR5 memory and is connected to the host through PCI express. Table 3 summarizes the systems used.

Program	Description	Problem Size (A)
BT	Dense linear algebra: use matrices/vectors to store data	64^3
EP	Embarrassingly parallel: low data dependency, low memory	2^{28}
FT	Spectral methods: fast Fourier transform	$256^2 X 128$
IS	Parallel sorting: bucket sort on integers	2^{23}
CG	Sparse linear algebra: data with many 0 values	14000
LU	Lower-Upper Gauss-Seidel solver	64^3

Table 2: Programs

B. Model Parameterization

1) Work Cycles per Instruction (WPI): To validate our hypothesis of constant WPI as workload scales from $\mathcal{P}_s$ to $\mathcal{P}$, we use hardware performance counters from PAPI to measure the cycles, instructions and determine WPI across

System	Host Intel Xeon E5-2680	MIC coprocessor Intel Xeon Phi 5110P
ISA	x86_64	x86_64
Cores	8	60
Threads/Core	2	4
Clock Frequency	2.7 GHz	1.053 GHz
L1 data cache	32kB / core	32kB / core
L2 cache	256kB	512kB
L3 cache	30MB	NA
Memory	128GB	8GB

Table 3: Intel node with MIC coprocessor

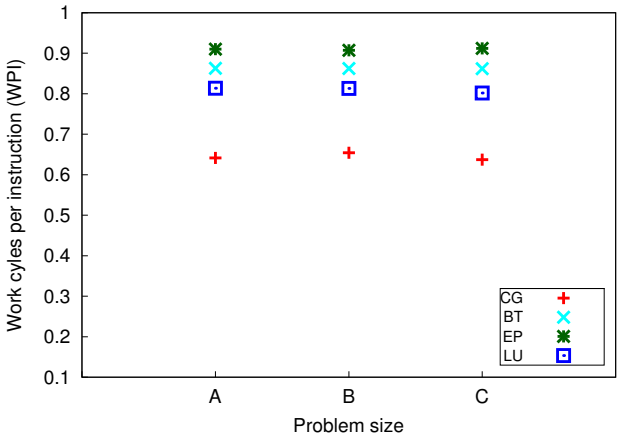

Figure 3: WPI validation

problem input sizes. Figure 3 plots[2] the WPI for problem sizes from A to C and shows that while the input sizes for the programs scale from 1 to 16 times, the WPI remains approximately constant. Thus, we measure WPI for class A and use it to predict the work cycles for program input sizes B and C using Equation 4. Next, we show the validation results of determining instructions for a larger problem size using a scaling factor.

2) Instruction Scaling: The proposed approach is practical because it uses the measured parameters by executing programs with smaller input sizes, and then determines time and energy-efficient configurations to execute the program with the scale-out problem size. Figure 4 plots measured versus predicted number of instructions for CG and FT with input size B using the measured values of class A. This validates the application of our approach and the usage of the scaling factor S to determine the useful work cycles using Equation 5.

3) Cache Accesses: The memory response time due intra- and inter-core contention depends on the number of requests to memory, model parameter, λ. Here, we show the validation of determining these model parameters for scale-out problem

[2]The WPI of the IS program for classes A, B and C is very close to EP and the WPI of the FT program for classes A and B is very close to CG. Thus these programs are not shown in the plot for better clarity. The FT program for input size C does not compile for the Xeon Phi system due to relocation overflows.

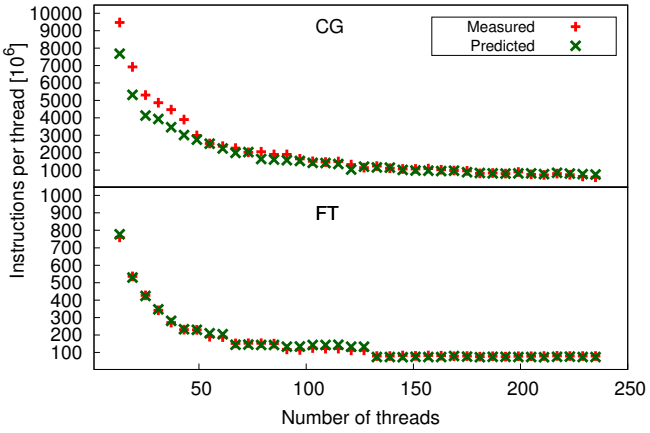

Figure 4: Validation of instruction scaling

sizes using the baseline measurements for smaller program size, $\mathcal{P}_s$. Figure 5[3] plots the measured versus predicted number of cache accesses for the programs FT and IS with input size as class B. The model predicts the class B values using the measurements from class A, which is four times smaller than class B. This validates the application of our approach and the usage of the scaling factor S to determine the model parameter, number of cache accesses used in Equations 7 and 10. While all the programs have a scaling factor of four, program CG accesses much more data in B compared to A and while the scaling factor for instructions is four, a separate scaling factor of 25 is used for data accesses.

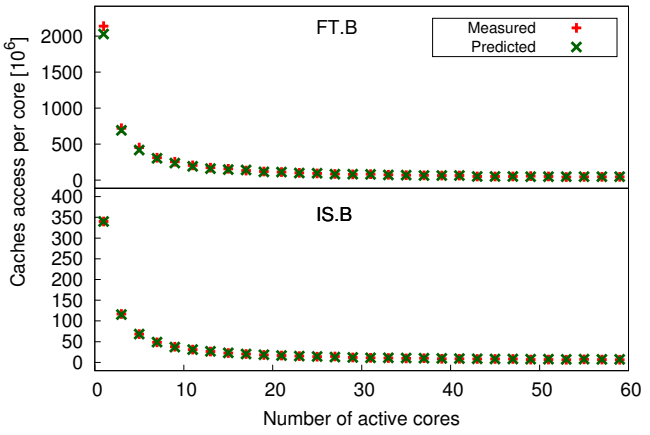

Figure 5: Validation of cache accesses

C. Execution time Validation

We use cycles as a handle for execution time. In this section we show the validation results for the cycles per thread for a single iteration of the program with problem size of class B. Extensive validation has been performed for all the programs in Table 2 across all possible configurations and the average error between the model and the measured values is shown in Table 4. Figure 6 and 7 show the validation results for the total cycles executed by programs LU, BT in scatter mode and programs FT, EP in compact mode respectively, as these programs have the worst average error.

[3]While Figures 4 and 5 show the plots for a subset of programs due to paucity of space, the results for all the programs are shown in Table 2.

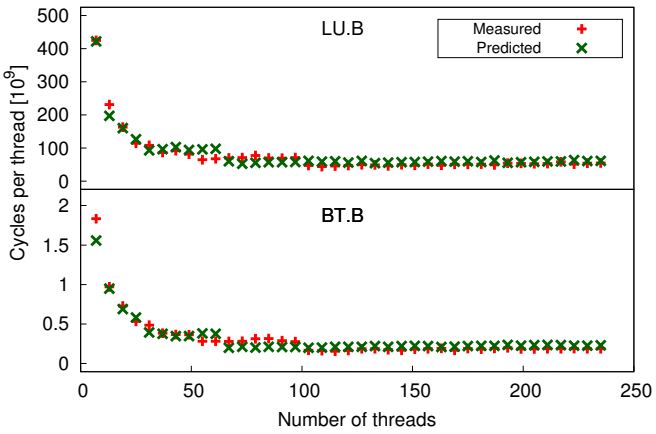

Figure 6: Validation results for scatter thread affinity mode

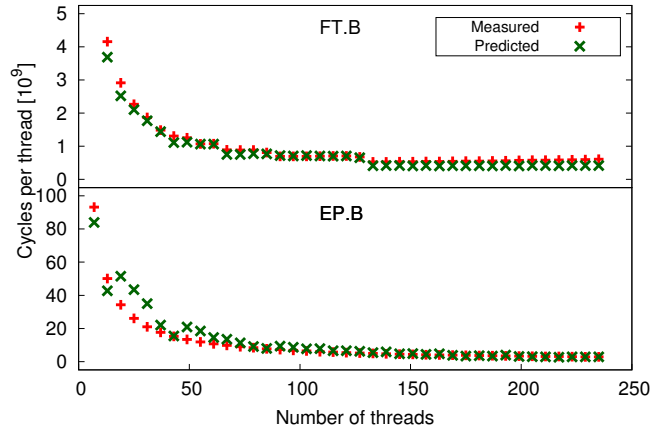

Figure 7: Validation results for compact thread affinity mode

Program	Average error [%]	
	Compact	Scatter
BT	7	8
EP	14	4
FT	15	8
IS	4	6
CG	1	6
LU	11	11

Table 4: Validation results

1) Sources of Inaccuracy: We identify three factors that affect the accuracy of the model. Firstly, the most significant source of error comes due to irregularities during different executions of the same program from the operating system

overheads. The measured values of execution time and energy show irregularities of up to 10% for different runs of the same program. Secondly, there are irregularities in both inter and intra-core contention for shared-memory due to the shared ring architecture as the shared data could be residing in either the neighbouring core or in the core that is half the distance of the ring from the current core. These irregularities vary the service time parameter of the memory and hence result in model errors. Thirdly, the model does not account for thread synchronizations and waiting time due to these barriers, which leads to loss of accuracy.

V. Analysis

In this section, we apply our model to study the energy efficiency of different configurations in MIC architecture system under a given service time deadline. We first show how our model can be applied to determine Pareto-optimal system configurations to execute a HPC program on MIC architecture system. Next, we present the performance-to-power ratio (PPR) of the MIC architecture system and determine the impact of PPR on the Pareto-optimal configurations.

A. Pareto-optimal Configurations

Similar to the Pareto frontiers in heterogeneous systems as reported in our earlier work [25], time-energy efficient Pareto-optimal configurations are also exhibited by MIC architecture systems executing HPC parallel programs as shown in Figures 8 and 9. These Pareto-optimal configurations are energy efficient as they consume the minimum energy for a given execution time deadline or execute in the minimum possible time for a given energy budget.

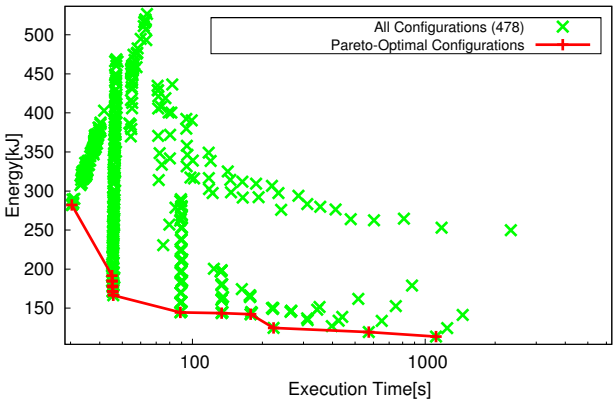

Figure 8: Pareto-optimal configurations for executing BT

Figures 8 and 9 present the execution time and energy used to execute parallel programs BT and FT respectively on all 477 possible configurations[4] of a MIC architecture system. Each of these configurations is a tuple consisting of the number of cores and the number of threads per core (c, τ), based on the thread affinity mode. Given an execution-time deadline, there exist a set of configurations that meet this deadline. The configuration that meets the deadline with the minimum energy usage is *Pareto optimal*. The set of all Pareto optimal points across all possible deadlines forms the time-energy *Pareto frontier*.

[4] $c \in [1, 2, 3, .., 60], \tau \in [1..4]$, mode $\in$ [scatter, compact], $60 \times 4 \times 2 = 480; c = 1, \tau = 1$; $c = 59$ with $\tau = 4$, & $c = 1$ with $\tau = 3$; and $c = 60, \tau = 4$ are the same in both modes, resulting in $480 - 3 = 477$ configurations.

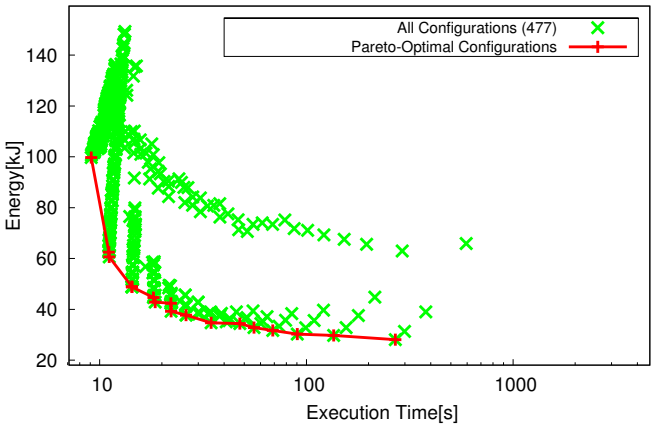

Figure 9: Pareto-optimal configurations for executing FT

Counter-to-intuition, as the execution-time deadline is relaxed, the configurations have lesser number of cores and threads but surprisingly use lesser energy. Reducing the number of cores decreases the average power used but increases execution time, and thus it is expected that the energy will be constant. Although reducing the number of cores causes a linear decrease in the power used, the effect on the execution time is non-linear and is characterized by the inter-core contention for shared resources such as memory (T_{inter}).

In the next section, we discuss the performance-to-power ratio metric and show how this metric is useful to determine the existence of Pareto-optimal configurations for a given program executing on accelerators with MIC architecture.

B. Performance-to-power Ratio

PPR is defined as the work done per unit of time, normalized by the average power consumption. This is equivalent to the work done per unit of energy and defined as,

$$PPR = \frac{Throughput[operations/s]}{Power[W]}$$

where throughput denotes the number of useful operations performed by the system per unit time[5]. This metric is also used in SPEC benchmark [29]. The floating point operations are used as the useful operations and the computed PPR for millions of floating point operations for all the programs considered in this paper across different number of active cores, for scatter and compact affinity modes with $\tau = 1$ and $\tau = 4$ respectively are shown in Table 5.

We observe that increasing the number of cores with $\tau = 1$ (scatter affinity) decreases the PPR, because execution time decreases sub-linearly while power increases linearly thus

[5] A bigger value is better

Program	Mega-operations (Mops)	PPR [Mops/J]							
		1 core		4 cores		16 cores		60 cores	
		$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 4$
BT	702,200	2.81	6.18	2.68	5.12	2.25	3.79	1.56	1.49
EP	2,147	0.06	0.14	0.06	0.13	0.06	0.11	0.06	0.07
FT	92,049	1.40	4.20	1.36	3.36	1.21	2.15	0.73	0.61
IS	335	0.10	0.20	0.09	0.20	0.08	0.18	0.07	0.08
CG	54,708	0.14	0.60	0.17	0.68	0.15	0.61	0.15	0.45
LU	498,816	1.83	4.20	1.67	3.36	1.39	2.26	1.06	0.83

Table 5: Performance-to-power ratio

resulting in an increase in energy, with the exception of the CG program. For the CG program, the execution time is much higher compared to the power of the Xeon Phi. Therefore, increasing the number of cores decreases the execution time by a larger ratio compared to the increase in power and thus reduces the total energy resulting in a better PPR.

For a given number of active cores, increasing the number of threads from one to four, improves the PPR as seen in Table 5. This is intuitive because the increase in power consumed by the core increases marginally, while the execution time decreases by at least a factor of two [7], [16]. This improvement in PPR is for smaller number of active cores, because as the number of active cores increases, the sequential fraction becomes the bottleneck, resulting in an increase in energy and thus reduction in PPR, as seen from results in Table 5 for $c = 60$.

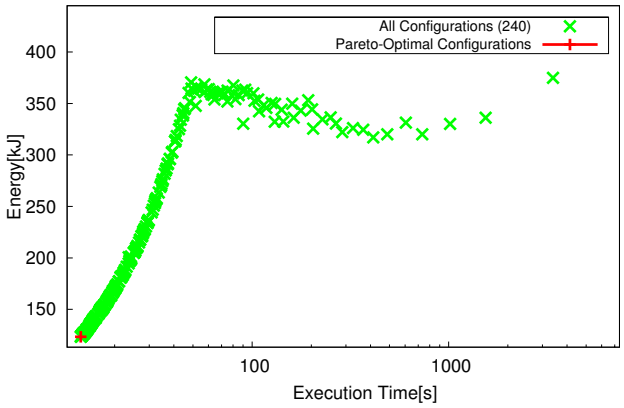

Figure 10: Pareto-optimal configurations for executing CG with scatter affinity mode

Next, we discuss the conditions under which a coprocessor with MIC architecture exhibits a Pareto frontier. To assess the impact of PPR, we analyse the Pareto-optimal configurations for the CG program as this program exhibited an improvement in PPR with increase in the number of active cores. Figures 10 and 11 plot the execution time and energy used across all possible configurations to execute the program CG with only scatter mode and both modes respectively. As observed from the plots, while there are many Pareto-optimal configurations for the CG program, when we consider only the scatter thread affinity mode, there is a single Pareto-optimal configuration. This scatter mode configuration uses 238 threads implying 58 cores with four active threads and two cores with three active threads.

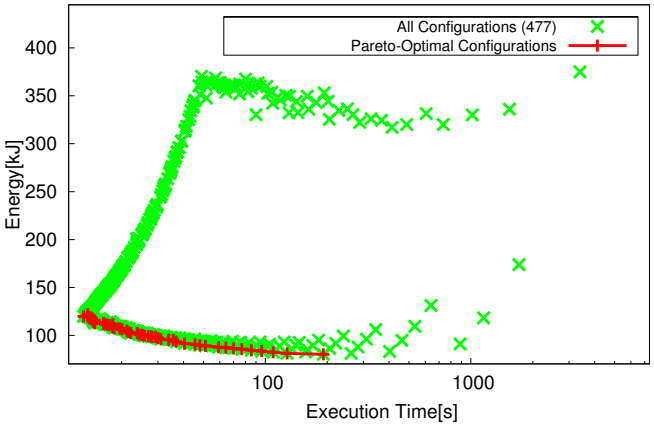

Figure 11: Pareto-optimal configurations for executing CG

The addition of the compact thread affinity mode exposes many more Pareto-optimal configurations as illustrated in Figure 11. These configurations are due to increasing the number of threads per core, with the number of active cores ranging from four to 60 cores, between the Pareto-configurations with the maximum and the minimum execution time. Thus, users of coprocessors with MIC architecture systems executing HPC applications can apply our approach to determine the thread affinity mode to use and the number of threads to execute their program with the best possible execution time and/or consuming the minimum possible energy.

a) Impact on Energy Savings: The existence of the Pareto frontier and using Pareto-optimal configurations for executing a program imply two options for saving energy. Firstly, for a given execution time deadline, using a Pareto-optimal configuration instead of a non-optimal configuration results in energy savings. For example, to execute the CG program within 17 seconds, while a non-optimal configuration with 192 threads[6] using scatter mode can consume up to 150kJ, a Pareto-optimal configuration with 187 threads[7] using compact mode consumes only 112kJ, thus resulting in energy savings of 25%.

Secondly, for configurations on the Pareto-frontier, significant energy savings can be obtained by trading-off execution time by a negligible amount. For example, to execute the BT program within 45 seconds, energy savings of 14% can be obtained at the expense of a 1% increase in execution time as illustrated in Figure 8.

[6]This configuration uses 12 cores with four threads per core and 48 cores core with three threads each.

[7]This configuration uses 46 cores with four threads per core and one core with one thread.

VI. Conclusions

While intra-node heterogeneous systems such as systems with MIC architecture coprocessors offer accelerated performance, users have to determine the time-energy optimal set of number of cores (c) and threads per core (τ) for executing the program in an energy-efficient manner. This paper presents an approach to determine time-energy Pareto-optimal system configurations (c, τ) for executing a parallel program on systems with MIC architecture using a measurement-driven analytical model.

The proposed model addresses the effects of both TLP within and across cores by considering *inter and intra-core resource overlaps*, *memory contention among threads within a core* and *contention across multiple cores*. Validation of the proposed approach for a range of HPC programs against direct measurement on Intel Xeon Phi coprocessor show an average error of up to 15% between the predicted and measured values.

We show that a Pareto frontier consisting of time-energy Pareto-optimal configurations exist for a parallel program executed on a MIC architecture system. These configurations either consume minimum energy for a given execution time deadline, or execute in the minimum possible time for a given energy budget. Hence, HPC users can easily apply our approach for time-energy efficient execution. To further understand the Pareto frontier, we use the performance-to-power ratio metric (PPR), that quantifies the amount of useful computations performed per unit energy used in an execution. Furthermore, we show the use-case of our approach to determine energy-efficient system configurations to execute programs on accelerators such as Xeon Phi and save energy.

VII. Acknowledgements

This work is supported by the National University of Singapore under grant number R-252-000-601-112.

References

[1] Top 500 supercomputer sites, online, 2015.

[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin et al, HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs, *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[3] T. Austin, E. Larson and D. Ernst, SimpleScalar: an Infrastructure for Computer System Modeling, *Computer*, 35(2):59–67, Feb 2002.

[4] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo et al, The NAS Parallel Benchmarks 2.0, Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[5] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski et al, A Regression-based Approach to Scalability Prediction, *Proc. of 22nd ICS*, pages 368–377, 2008.

[6] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi et al, The M5 Simulator: Modeling Networked Systems, *Micro, IEEE*, 26(4):52–60, July 2006.

[7] S. Borkar, Thousand core chips: a technology perspective, *Proceedings of the 44th DAC*, pages 746–749, 2007.

[8] I.-H. Chung, S. R. Seelam, B. Mohr and J. Labarta, Tools for Scalable Performance Analysis on Petascale Systems, *Proc. of IPDPS*, pages 1–3, May 2009.

[9] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P.-K. Yeung et al, On the Communication Complexity of 3D FFTs and its Implications for Exascale, *Proc. of 26th ICS*, pages 205–214, 2012.

[10] K. Czechowski and R. Vuduc, A Theoretical Framework for Algorithm-architecture Co-design, *Proc. of 27th IPDPS*, pages 791–802, 2013.

[11] J. Fang, A. L. Varbanescu, H. Sips, L. Zhang, Y. Che et al, An Empirical Study of Intel Xeon Phi, *arXiv preprint arXiv:1310.5842*, 2013.

[12] J. Fang, A. L. Varbanescu, H. Sips, L. Zhang, Y. Che et al, Benchmarking Intel Xeon Phi to Guide Kernel Design, *Delft University of Technology Parallel and Distributed Systems Report Series, PDS-2013-005*, 2013.

[13] C. George and Intel, Intel Xeon Phi X100 Family Coprocessor - the Architecture, *https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner*, November 2012.

[14] J. Guo, J. Meng, Q. Yi, V. Morozov and K. Kumaran, Analytically Modeling Application Execution for Software-Hardware Co-design, *Proc of 28th IPDPS*, pages 468–477, 2014.

[15] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov et al, Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems based on Intel® Xeon Phi Coprocessor, *Proc. of 27th IPDPS*, pages 126–137, 2013.

[16] M. Hill and M. Marty, Amdahl's Law in the Multicore Era, *Computer*, 41(7):33–38, 2008.

[17] C. L. Janssen, H. Adalsteinsson and J. P. Kenny, Using Simulation to Design Extremescale Applications and Architectures: Programming Model Exploration, *SIGMETRICS Perform. Eval. Rev.*, 38(4):4–8, Mar. 2011.

[18] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin et al, Compiler-based data prefetching and streaming non-temporal store generation for the intel (r) xeon phi (tm) coprocessor, *Proc. of 27th IPDPSW*, pages 1575–1586, 2013.

[19] B. C. Lee and D. M. Brooks, Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction, *In Proc. of 12th ASPLOS*, volume 41, pages 185–194, 2006.

[20] G. Mao, D. Böhme, M.-A. Hermanns, M. Geimer, D. Lorenz et al, Catching Idlers with Ease: A Lightweight Wait-State Profiler for MPI Programs, *Proc. of the 21st EuroMPI*, Sept. 2014.

[21] H. McCraw, J. Ralph, A. Danalis and J. Dongarra, Power monitoring with papi for extreme scale architectures and dataflow-based programming models, *Proc. of CLUSTER*, pages 385–391, 2014.

[22] A. Morris, A. Malony, S. Shende and K. Huck, Design and Implementation of a Hybrid Parallel Performance Measurement System, *Proc. of 39th ICPP*, pages 492–501, Sept 2010.

[23] A. Ramachandran, J. Vienne, R. Van Der Wijngaart, L. Koesterke and I. Sharapov, Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi, *Proc. of 42nd ICPP*, pages 736–743, 2013.

[24] L. Ramapantulu, D. Loghin and Y. M. Teo, An Approach for Energy Efficient Execution of Hybrid Parallel Programs, *Proc. of IPDPS*, pages 1000–1009, 2015.

[25] L. Ramapantulu, B. M. Tudor, D. Loghin, T. Vu and Y. M. Teo, Modeling the Energy Efficiency of Heterogeneous Clusters, *Proc. of 43rd ICPP*, pages 321–330, 2014.

[26] S. Ramos and T. Hoefler, Modeling Communication in Cache-coherent SMP systems: A Case-study with Xeon Phi, *Proc. of 22nd HPDC*, pages 97–108, 2013.

[27] D. Schmidl, T. Cramer, S. Wienke, C. Terboven and M. S. Müller, Assessing the Performance of OpenMP Programs on the Intel Xeon Phi, *Proc. of Euro-Par*, pages 547–558. Springer, 2013.

[28] M. Si, A. J. Peña, P. Balaji, M. Takagi and Y. Ishikawa, MT-MPI: Multithreaded MPI for Many-core Environments, *Proc. of the 28th ICS*, pages 125–134, 2014.

[29] SPEC, SPEC Power and Performance, Benchmark Methodology V2.1, *https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf*, 2011.

[30] V. Taylor, X. Wu and R. Stevens, Prophesy: an Infrastructure for Performance Analysis and Modeling of Parallel and Grid applications, *SIGMETRICS Perf. Eval. Review*, 30(4):13–18, 2003.

[31] B. M. Tudor and Y. M. Teo, On Understanding the Energy Consumption of ARM-based Multicore Servers, *Proc. of SIGMETRICS*, pages 267–278, 2013.

[32] R. F. Van der Wijngaart and H. Jin, NAS Parallel Benchmarks, Multi-zone Versions, 2003.

[33] A. Vladimirov and V. Karpusenko, Heterogeneous Clustering with Homogeneous Code: Accelerate MPI Applications without Code Surgery using Intel Xeon Phi coprocessors, *Colfax White Paper*, 2013.

[34] V. M. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula et al, Papi 5: Measuring power, energy, and the cloud, *Proc. of ISPASS*, pages 124–125, 2013.