2015 IEEE 29th International Parallel and Distributed Processing Symposium

An Approach for Energy Efficient Execution of Hybrid Parallel Programs

Lavanya Ramapantulu, Dumitrel Loghin 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

***Abstract*—Hybrid programming model is becoming increasingly popular for HPC applications as it has the dual-advantage of exploiting inter-node distributed-memory scalability and intra-node shared-memory performance in a cluster system. One of the key challenges for energy efficient execution of hybrid programs is to determine time and energy efficient hardware configurations among a large system configuration space. Given a hybrid 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-node resource overlaps, memory contention among cores within a node and network contention across multiple nodes. The model is validated against direct measurement using five representative HPC applications on Intel Xeon and ARM clusters having diverse time-energy performance. We show that a Pareto frontier consisting of optimal configurations exist for a hybrid program running on homogeneous clusters. To further optimize the Pareto frontier, we introduce a new metric, useful computation ratio (UCR) to quantify the degree of resource contentions and communication overheads in an execution. We discuss how UCR and Pareto-optimal configurations can be used in conjunction by system's designers to gain further insights into system resource imbalances, and how application developers can further fine-tune their hybrid programs.**

I. Introduction

With the advent of dark silicon era limiting the thermal and power budgets of a multi-core processing node [19], distributed systems are the way forward. However, it is well known that High Performance Computing (HPC) applications are limited by data access bottlenecks. To simultaneously exploit the computational capabilities of multi-core systems and to scale complex HPC applications, application developers are increasingly using hybrid programming models. A hybrid programming model utilizes both distributed-memory across nodes for scalability and shared-memory within a node for improving performance [6], [31], [43].

A hybrid parallel program is partitioned into a variable number of logical parallel processes and parallel threads. For a given hybrid program and a multi-node system with multi-core nodes operating at different core clock frequencies, there is a large system configuration space for executing these logical processes and threads. As the resource demands in a hybrid parallel program varies with its problem size, these resource demands have to be mapped onto different system configurations to minimize resource contention and runtime overheads. Hence, energy-efficient execution of hybrid programs is non-trivial and poses a number of research challenges such as:

1) What is an energy efficient system configuration (number of nodes, number of cores and core clock frequency) to execute a hybrid program?
2) How much time does a hybrid program spend on computation (useful work) versus communication of data and other overheads, and what is a good Useful Computation Ratio (UCR)?
3) Does a higher UCR imply a more energy-efficient configuration for executing a hybrid program?

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 energy-efficiency. This paper addresses these challenges and proposes an approach to execute a hybrid (MPI+OpenMP) parallel program in an energy efficient manner. An energy efficient execution configuration uses minimum energy to meet a given execution time deadline.

Current approaches to analyze hybrid 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 [3], [12], [33], [37]. 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 [5], [8], [26]. However, analyzing application executions even with reasonable input sizes is very time-consuming [22].

This paper presents an approach to determine time and energy efficient system configurations for executing a hybrid 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 novelties of our approach are modeling both inter and intra-node resource overlaps and resource contention. Given a hybrid program, the proposed approach determines energy-efficient Pareto-optimal configurations in terms of the number of nodes, number of cores per node and core clock frequency. These configurations either consume minimum energy for a given

1530-2075/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPS.2015.71

execution time deadline[1], or execute in the minimum possible time for a given energy budget. Thus, providing a systematic method to set the number of logical processes and threads for efficient execution of a hybrid program. Secondly, to quantify the degrees of resource contention and communication overhead in an execution, we introduce the Useful Computation Ratio (UCR) metric. We also discuss how UCR and Pareto-optimal configurations can be used in conjunction by system designers to gain further insights into resource imbalances and how application developers can fine-tune their hybrid program.

The proposed model is validated against direct measurements on both ARM Cortex-A9 and Intel Xeon based clusters using a range of HPC applications including nonlinear partial differential equation solvers [50], electronic-structure calculations, nanoscale materials modeling [21] and simulation of computational fluid dynamics [23]. 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 time-energy Pareto-optimal configurations for HPC applications and discuss the use of UCR to analyze and optimize these configurations further.
Our key contributions are:

1) A measurement-driven analytical model to determine time-energy performance of a hybrid parallel program. In contrast to current approaches, we model both inter and intra-node resource overlaps, memory contention among cores within a node and network contention across multiple nodes.
2) A new approach that determines time-energy Pareto-optimal system configurations to execute a hybrid program with a given time deadline and energy budget. We show that a Pareto frontier consisting of optimal configurations exist for a hybrid program executed on a homogeneous cluster of multicore nodes.
3) A new normalised metric, useful computation ratio (UCR) that quantifies the degree of runtime resource contentions and communication overheads. We show how the UCRs of Pareto-optimal configurations are increased by balancing resource service demands with resource utilization, to further minimize system inefficiencies.

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) approaches to improve energy efficiency and (ii) performance modeling of hybrid programs, and compare and contrast these with our proposed approach.

[1]While most HPC applications do not have strict deadlines, their execution times are constrained due to sharing of cluster resources. Also, with the advent of pay-per-use models, an execution time deadline translates to a cost budget.

A. Energy efficient execution

A commonly used technique to improve energy efficiency is Dynamic Voltage and Frequency Scaling (DVFS) [20], [25], [27], [36]. For MPI and hybrid OpenMP+MPI based iterative programs, such DVFS based techniques exploit the inter-node slack, by lowering the frequency/voltage of nodes that are idling at synchronization points in the program. As these approaches are applicable at run-time in a dynamic manner, they can be used in conjunction with our proposed approach.

Another approach used along with DVFS is Dynamic Concurrency Throttling (DCT). There are many state-of-the-art algorithmic strategies using combined DCT and DVFS techniques to save energy without losing execution time performance. While Curtis-Maury et al. [13], [14] use DVFS and DCT for OpenMP programs, the algorithms proposed by Dong et al. [29] incorporate MPI communication to OpenMP phases and hence is applicable to hybrid programs as well. While these methods use multivariate linear regression to obtain model coefficients, we use an analytical model based on measured parameters to derive architectural artefacts.

There are several algorithmic first-principle approaches to improve energy efficiency by obtaining complexity bounds on time-energy performance of applications using roofline models [10], [11], [17], [53]. However, these approaches optimize energy efficiency at the algorithmic level, while our approach determines Pareto-optimal hardware configurations to execute a given application. While there are simple and fundamental formulae that describe the interplay between program parallelism, speedup and energy consumption [9], [24], [55], this work uses measurements to derive inputs to the analytical expressions and hence is more accurate. Other approaches to improve the energy efficiency of program executions involve dynamic scheduling of processes on heterogeneous cores within a node [30], [38], [49]. These methods complement our approach as we determine Pareto-optimal configurations based on a measurement-driven analytical model.
In contrast to current methods, our approach exploits the large configuration space and determines Pareto-optimal configurations to achieve energy-efficient hybrid program execution.

B. Performance modeling

More recently, at an algorithm level, asymptotic analysis based modeling techniques are used to derive trade-offs between computation and communication [15], [16]. 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 is our previous work which use non-intrusive inputs from operating system traces and hardware event counters [40], [46], [47]. While these works focus on parallel programs, they do not consider communication overheads. This paper focuses on hybrid parallel programs and models both inter-node communication, and resource contention. Other alternative approaches to predict performance include statistical methods that rely on black-box regression to infer dependencies between hardware parameters

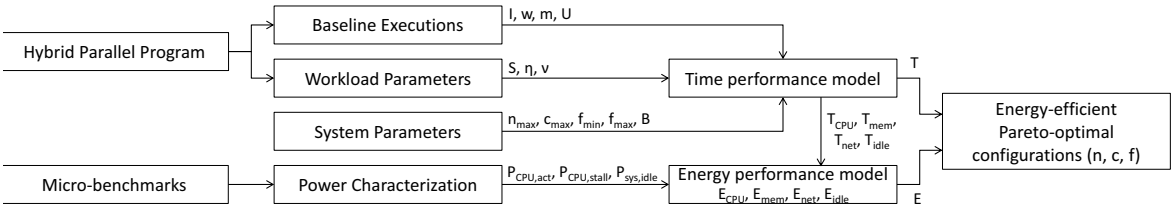

Figure 2: Approach overview

and application performance [7], [28], [45]. Our approach is not black-boxed and thus enables analytical prediction of the impact of changing different system components such as memory/network bandwidth on program execution time performance.

III. Proposed Approach

In this section, we discuss our proposed analytical model. We first present an overview of our model and its assumptions. Next, we derive the time-energy performance model followed by its input.

A. Overview

A typical hybrid parallel program is split into ℓ logical processes with τ parallel threads per process as shown in Figure 1. Systems executing hybrid programs have a peak power budget that limits the number of nodes to a maximum of n_{max}. Each node can be configured to use a maximum number of c_{max} cores, with each core operating at a clock frequency $f \in [f_{min}, f_{max}]$. Hence, the different combinations of n, c, and f values within these bounds form the total number of system configurations for executing the program. Users of hybrid programs often face the challenge of determining the optimal number of ℓ and τ for energy efficient execution. Using values of $\ell < n$ incurs energy wastage due to idling resources. On the contrary, it may be beneficial to have $\tau < c_{max}$ because of the energy saved by reducing the waiting time due to shared-memory contention among the τ threads. Hence, choosing an energy efficient number of ℓ (= n) and τ (= c) to execute a hybrid program is not obvious and requires an approach that models the execution time considering inter and intra-node (i) overlap, (ii) communication, and (iii) contention for shared resources. Therefore, as outlined in Figure 2, given a hybrid parallel program, our approach uses a measurement-driven analytical model to determine Pareto-optimal system configurations, i.e. (n, c, f), such that these configurations consume minimum energy for a given execution time deadline and/or execute in the minimum time for a given energy budget.

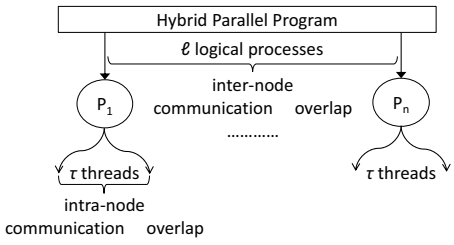

Figure 1: Model of hybrid program execution

Symbol	Description
Workload Parameters	
$\mathcal{P}$	hybrid program
$\mathcal{P}_s$	$\mathcal{P}$ with smaller input size
S	no. of iterations in $\mathcal{P}$
S_s	no. of iterations in $\mathcal{P}_s$
η	no. of messages sent/received by $\mathcal{P}$
ν	volume (in bytes) per message
Baseline Execution	
I_s	no. of instructions in $\mathcal{P}_s$
w_s	no. of work cycles in $\mathcal{P}_s$
b_s	no. of non-memory stall cycles in $\mathcal{P}_s$
m_s	no. of memory-related stall cycles in $\mathcal{P}_s$
U_s	CPU utilization for $\mathcal{P}_s$
System Parameters	
n_{max}	maximum number of nodes
c_{max}	maximum number of cores per node
f_{max}	maximum core clock frequency
B	communication throughput
Power Parameters[W]	
$P_{CPU,act}$	CPU power when executing work cycles
$P_{CPU,stall}$	CPU power during memory-related stalls
P_{mem}	Power consumed by memory operations
P_{net}	Power consumed by network card
$P_{sys,idle}$	power consumption of idling system
Time Model	
w	work cycles for $\mathcal{P}$
b	non-memory stall cycles for $\mathcal{P}$
m	memory-related stall cycles for $\mathcal{P}$
U	CPU utilization for $\mathcal{P}$
n	number of nodes to execute $\mathcal{P}$
c	number of cores per node that execute $\mathcal{P}$
f	operating core clock frequency
T_{CPU}	time for computation
$T_{w,mem}$	waiting time due to memory contention
$T_{s,mem}$	non-overlapped memory service time
$T_{w,net}$	waiting time due to network contention
$T_{s,net}$	non-overlapped network service time
T	total execution time of a program
Energy Parameters[kJ]	
$E_{CPU,act}$	energy consumed when CPU is active
$E_{CPU,stall}$	energy consumed when CPU is stalling
E_{mem}	energy consumed by memory sub-system
E_{net}	energy consumed by network sub-system
E_{idle}	energy consumed by idle system
E	total energy consumed by a program

Table 1: Model parameters

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 on a single node across all possible c and f values using a small program input size. To derive the energy consumed by the program, we use micro-benchmarks to measure the power characteristics of the processor system.

While our previous work models the overlap between inter-node computation phases, it considers data center workloads that have minimal inter-node communication [40]. However, hybrid programs not only exhibit considerable amount of inter-node communication among ℓ processes but also intra-node communication among τ threads. Hence modeling the overlaps, communication and contention for shared resources for a hybrid program is non-trivial and challenging. To determine the inter-node communication, we use the volume of communication per node (ν) and characterize the CPU overheads in communication using the utilization of the cores (U_s). This models the overlap between computation and inter-node communication via the network. Next, we use queueing theory to model contention among the nodes for network and compute the waiting time ($T_{w,net}$) at each node with the network switch as a server servicing communication requests. To model the overlap between computation and intra-node communication via shared-memory, we infer the waiting time of the cores due to contention for memory using stall cycles (m). The notations used in our approach are described in Table 1.

B. Assumptions

An abstract view of a typical hybrid parallel program can be represented as iterations of alternating computation and communication phases. While the computation phase is further split into parallel threads performing computations using shared-memory data, the communication phase consists of logical processes on different nodes using MPI over the network. We assume that these phases have negligible resource demands from storage devices such as disks. The code in Listing 1 shows an abstraction of a hybrid parallel program [39] with annotations illustrating the contention for shared-memory and network as inferred by our approach.

Listing 1: Abstraction of a Hybrid program

```
for(iteration = 1..S)
{
  # pragma omp parallel // τ threads on c cores
  {
        /* computations or useful work */
        ....
        ....
        ....
        /* intra-node (shared-memory contention) */
  }
  /* inter-node (network contention) */
  MPI_Send // ℓ logical processes on n nodes
  MPI_Recv
}
```

To show that our approach is independent of a programming language, we have chosen benchmark applications in both C++ and Fortran. The five hybrid programs are, Block Tri-diagonal solver (BT), Lower-Upper Gauss-Seidel solver (LU), Scalar Penta-diagonal solver (SP) [50], Car-Parrinello Molecular Dynamics (CP) [21] and a lattice Boltzmann method (LB) [23]. To predict the energy usage for a configuration, it is assumed that the hybrid program is the only application being executed, apart from background operating system tasks.

To demonstrate the application of our approach on different system architectures, we validated it on two systems with diverse time-energy performance, ARM Cortex-A9 processor based low-power cluster and Intel Xeon $x86\_64$ architecture based processor cluster. The systems used for validation have nodes with a single NIC and cores access shared-memory via Uniform Memory Architecture (UMA). Nodes communicate via an Ethernet-based switch.

C. Time Performance Model

In this section, we derive the execution time, T, for a hybrid program $\mathcal{P}$ with S iterations, executing on n homogeneous nodes, each having c cores. Inter and intra-node overlap between the computation and communication phases in the program is accounted as useful work cycles (T_{CPU}). Non-overlapped execution time including both inter and intra-node data dependencies is modeled as waiting and service time for communication over the network ($T_{w,net}, T_{s,net}$) and within shared-memory ($T_{w,mem}, T_{s,mem}$) respectively. Hence, the total execution time is summed up as the overlapped time for useful work cycles and the non-overlapped queueing delays:

$$T = T_{CPU} + T_{w,net} + T_{s,net} + T_{w,mem} + T_{s,mem} \quad (1)$$

To derive T_{CPU}, we use the total work cycles incurred by the hybrid program. The total work cycles for a given program is split equally among number of processing cores across all nodes, operating at a clock frequency f as:

$$T_{CPU} = \frac{cycles_{core}}{n \cdot c \cdot f} \quad (2)$$

Overlap of computations and shared-memory data accesses is accounted by using the work cycles spent for computing (w) and stall cycles that are not due to memory contention (b). These non-memory stalls are due to the complex out-of-order pipeline architectures that are prevalent in most processors today [34]. Hence, the total useful cycles considering overlaps is:

$$cycles_{core} = w + b \quad (3)$$

As non-memory stalls vary based on the application, they are measured using the baseline execution of a program, but they scale well with program input size for a given processor architecture [40]. Hence, they are derived easily for scale-out programs, by using the measured values of the subset program $\mathcal{P}_s$:

$$w(c,f) = \frac{w_s \cdot S}{S_s} \qquad b(c,f) = \frac{b_s \cdot S}{S_s} \quad (4)$$

While parallelism increases speedup by enabling overlap, overlap causes increased contention for shared resources. For hybrid parallel programs, the logical processes across nodes contend for access to the network and the parallel threads within a logical process contend for shared-memory. Network contention causes messages to wait in the operating system's network socket buffer before being serviced by the network. As a result, CPU idles while waiting for data from the network.

This is modeled using a M/G/1 queue with a mean waiting time [35], [41]:

$$T_{w,net} = \frac{\lambda \cdot \hat{y}^2}{1-\rho} \tag{5}$$

where $\hat{y}$ and ρ are the service time and utilization of the network respectively, and λ is the inter-arrival rate of messages to the buffer.

Communication characteristics of the program affect inter-node communication ($T_{s,net}$). For hybrid parallel programs, the communication characteristics are determined using the number of messages transmitted (η) and the volume of communication per message (ν). In most modern processing systems, the CPU time incurred for processing, overlaps with transfer time of the messages over the network. Thus, the service time of inter-node communication is:

$$T_{s,net} = \max\left(((1-U)\cdot T_{CPU}), \left(\frac{\eta \cdot \nu}{B}\right)\right) \tag{6}$$

In hybrid programs, the parallel threads within a logical process contend for shared-memory. Shared-memory contention is derived from the waiting time and service time of memory requests queueing up for service at the memory controller. Queueing delay due to memory contention causes stall cycles in the processor. Therefore, these stall cycles due to non-overlapped memory accesses (m) are used to model shared-memory contention overheads:

$$T_{w,mem} + T_{s,mem} = \frac{m}{f} \qquad m(c,f) = \frac{m_s \cdot S}{S_s} \tag{7}$$

D. Energy Performance Model

Total energy for a given hybrid program on a cluster of n nodes is the sum of the energies consumed per node. Energy consumed by a node is divided among three active components, processing unit (CPU), memory resources and network card. Energy consumed by other system components such as power regulators, storage, video, etc. are considered under the E_{idle}. Hence, the total energy consumed by the system during execution is:

$$E = \left(E_{CPU} + E_{mem} + E_{net} + E_{idle}\right) \times n \tag{8}$$

Energy consumed by the active cores in a node is:

$$\begin{aligned} E_{CPU} =& ((P_{core,act} \cdot T_{CPU}) + \\ & (P_{core,stall} \cdot (T_{w,mem} + T_{s,mem}))) \cdot c \end{aligned} \tag{9}$$

Energy consumed by the memory and network for each node is:

$$E_{mem} = P_{mem} \cdot (T_{w,mem} + T_{s,mem}) \tag{10}$$

$$E_{net} = P_{net} \cdot (T_{w,net} + T_{s,net}) \tag{11}$$

When the system is completely idle, the power consumption includes the idle power of the cores, memory and I/O devices, as well as the fixed power consumption for the rest of the components. Thus,

$$E_{idle} = P_{sys,idle} \cdot T \tag{12}$$

E. Model Inputs

The measurement driven inputs to our analytical model are obtained from workload, network and power characterization.

1) Workload Characterization: To derive workload dependent architectural artefacts, we use baseline executions of the program on a single node. To determine the overlap among useful computation cycles, data-accesses from shared-memory and network, we measure the translation of a given hybrid program into useful work cycles (w_s). To model the non-overlapped intra-node contention, we measure the stall cycles due to memory accesses (m_s). These measurements are recorded for a single node across the possible values of c and f using hardware performance counters. Hence, these measurements are non-intrusive with respect to the execution of the application. Program dependent communication characteristics, such as number of communication calls (η) and communication volume per message call (ν) are measured using the lightweight profiling tool mpiP [51]. It suffices to perform baseline executions only on a single node, as workload characteristics from these measured values can be inferred from ℓ and τ.

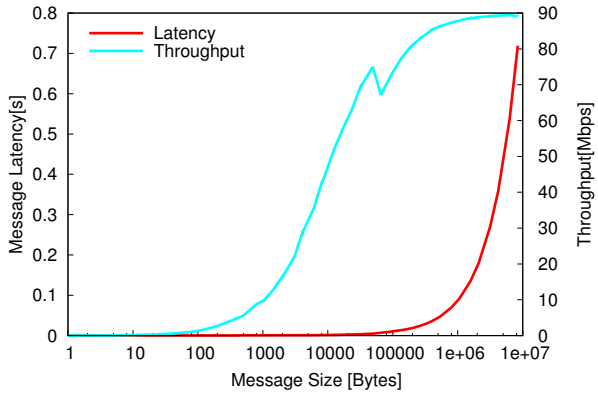

Figure 3: Network characterization

2) Network characterization: To measure communication overheads of MPI over TCP for a given link bandwidth, we use NetPIPE [48]. Figure 3 shows that the maximum achievable throughput on a 100 Mbps Ethernet link is only 90 Mbps due to MPI overheads and the operating system. This characterization of the network link latency and bandwidth is used to compute network service time.

3) Power Characterization: With energy proportionality becoming increasingly important, processors exhibit a wide dynamic energy range [54]. Hence, during program execution, cores have a wide-ranging power consumption depending on the amount of computations being executed. We classify the power states of a core into active power and stall power corresponding to computation cycles and stall cycles respectively. The idle power of the core is accounted by the total system idle power ($P_{sys,idle}$). We developed benchmarks that stress the processor pipeline to measure active and stall CPU power. These measurements are done for the complete range

of cores (c) and frequencies (f) supported by the system to characterize the processor across its dynamic power range. We derive P_{mem} from JEDEC memory specifications and directly measure P_{net}.

IV. Validation

A. Validation Results

Here, we first describe the hybrid programs, systems and setup used for validation followed by validation results. The proposed approach is validated against direct measurements for both execution time and energy.

B. Workloads and Setup

While our approach is applicable on generic hybrid parallel programs, we selected a representative subset of five benchmark programs for presentation in this paper. This subset was chosen to represent a wide range of HPC domain applications that exert different inter and intra-node communication resource demands and use different programming languages. We use three hybrid programs from NASA Parallel Benchmark (NPB) suite [50]. These solve discretized version of Navier-Stokes equations in three dimensions, and are (i) Lower-Upper Symmetric Gauss-Seidel (LU), (i) Scalar Penta-diagonal (SP), and (iii) Block Tri-diagonal (BT). The fourth program uses the Car-Parinello (CP) method to simulate H_2O molecules from the Quantum Espresso suite [21]. While the above programs are in Fortran, we chose the fifth program in C++, to illustrate that our approach is independent of the programming language. This is an open source Lattice Boltzmann (LB) code [1], that simulates fluid flows in a three-dimensional lid-driven cavity.

System	Intel Xeon E5-2603	ARM Cortex-A9
ISA	x86_64	ARMv7-A
Nodes	8	8
Cores/node	8	4
Clock Frequency	1.2–1.8 GHz	0.2–1.4 GHz
L1 data cache	32kB / core	32kB / core
L2 cache	2MB / node	1MB / node
L3 cache	20MB / node	NA
Memory	8GB DDR3	1GB LP-DDR2
I/O bandwidth	1Gbps	100Mbps

Table 3: Systems used for validation

To illustrate the generalization of our approach, we validate on two diverse processor system clusters as detailed in Table 3. The increase in the computing capabilities of mobile-based smart devices, have caught the attention of leading server providers to design their next-generation systems based on such mobile-based processors [4], [42], [44]. Hence, other than the traditional server system based on two Intel Xeon CPUs, we also choose a low-power ARM Cortex-A9-based system for validating the proposed approach. The Xeon and ARM systems not only have a diverse performance-to-power ratio but also have different ISAs and differ in orders of magnitude in their cache, memory and network bandwidths. These large differences among the resource capabilities of the selected systems illustrates that our approach can be applied to a generic processor system and is independent of any specific

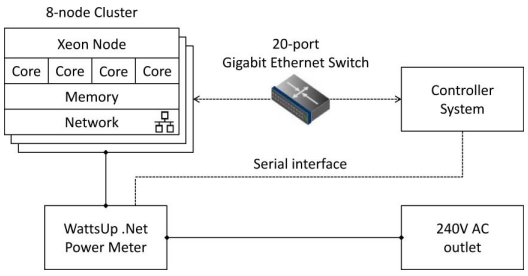

Figure 4: Validation setup

ISA. We validate our model against direct measurements of both execution time and energy usage of each cluster with the setup[2] shown in Figure 4. The system time command is used to measure execution time and a WattsUp meter [2] measures both power and energy.

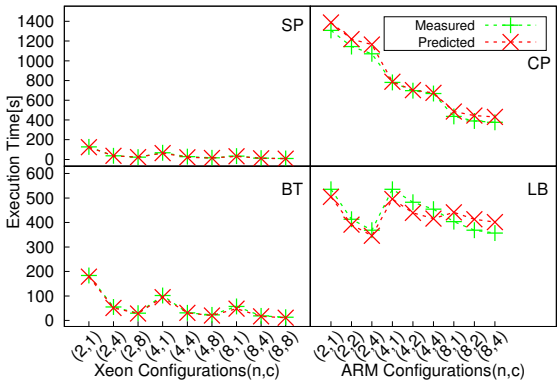

Figure 5: Execution time validation

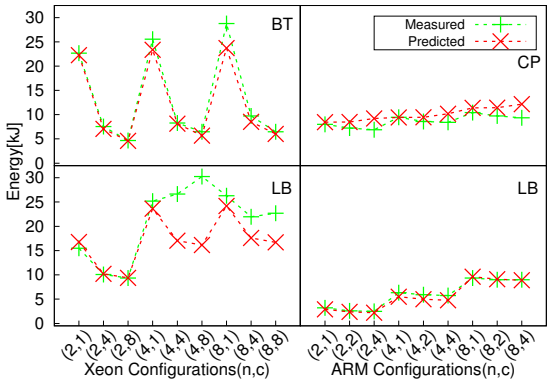

Figure 6: Energy validation

To increase the credibility of our approach, we performed extensive validation for each of the five benchmarks on a large number of Xeon and ARM system configurations. These configurations arise from varying the (i) number of nodes, $n_{xeon/arm} \in [1, 2, 4, 8]$, (ii) number of active cores per node, $c_{xeon} \in [1 \cdots 8]$ and $c_{arm} \in [1 \cdots 4]$), and (iii) operating core clock frequency, $f_{xeon} \in [1.2, 1.5, 1.8]$ GHz and $f_{arm} \in [0.2, 0.5, 0.8, 1.1, 1.4]$ GHz. Thus, the number of

[2]Figure 4 shows the Xeon cluster. The ARM cluster has a similar setup.

Domain	Benchmark Suite	Program	Execution Time error[%]				Energy error[%]			
			Xeon		Cortex-A9		Xeon		Cortex-A9	
			Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
3D Navier-Stokes Equation Solver	NAS Multi-zone Parallel Benchmark (NPB3.3-MZ)	LU	4	5	3	2	5	8	6	6
		SP	6	9	4	3	2	10	4	5
		BT	8	7	4	6	8	7	5	6
Electronic-structure Calculations	Quantum Espresso (v5.1)	CP	1	10	5	12	1	14	7	12
Computational Fluid Dynamics	OpenLB (olb-0.8r0)	LB	6	8	4	8	15	12	7	9

Table 2: Cluster validation results

configurations used for validation were 96 and 80 for Xeon and ARM clusters respectively. Table 2 summarizes the average error and the standard deviation from the measured values for all of these configurations. The predicted values of time and energy using our approach follow the trends of the measured values across hardware configurations as shown in Figures 5 and 6 respectively. Due to paucity of space, for each cluster we plot the execution time and energy for programs with the worst-case error.

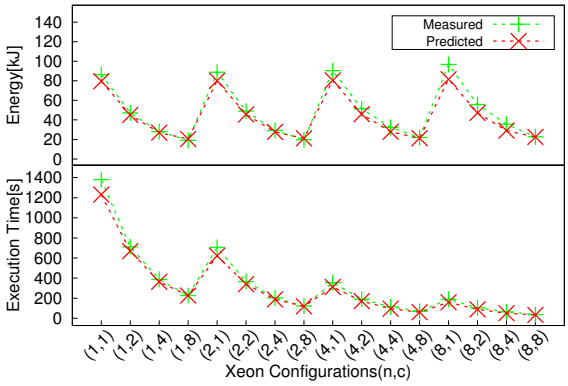

Figure 7: Scale-out program LU

Many research studies show a correlation between communication patterns exhibited by a program with scale-out input sizes [32], [52]. We show the application of our approach for scale-out HPC programs by plotting the validation results for LU program with input size of class C (four times larger than the baseline measurement program size) in Figure 7. Thus, we show the application of our model to programs whose communication characteristics scale linearly with respect to program input size.

C. 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 the communication overheads due to explicit synchronizations in the program among logical processes and threads. For example, LB program incurs more instructions on higher number of nodes at higher number of cores, due to the synchronization among the logical processes and threads. This significantly increases the energy used, but does not reduce the execution time. This increase causes our model to underestimate the energy used by Xeon configurations (4,4) and (4,8) as shown in Figure 6. The third reason for model inaccuracy is the accuracy of the characterized power parameters. In particular, the system power values for active cycles, stall cycles and idleness differ by up to $0.4W$ for the ARM node and $2W$ for the Xeon node. This variability translates into a larger underestimation of the energy consumed especially for larger execution times.

V. Energy Efficiency Analysis

This section discusses the application of our approach to determine time-energy Pareto-optimal configurations for efficient execution of hybrid parallel programs. Next we discuss the application of the Useful Computation Ratio (UCR) metric to further optimize the Pareto frontier.

A. Pareto-Optimal Configurations

Similar to the Pareto frontiers in heterogeneous systems as reported in our earlier work [40], time-energy efficient Pareto-optimal configurations are also present in homogeneous systems executing hybrid 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. Figures 8 and 9

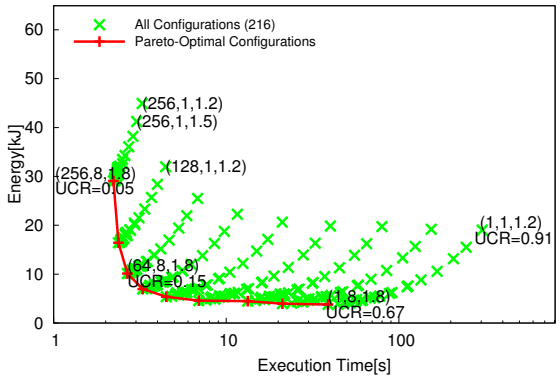

Figure 8: Xeon cluster executing SP program

present two typical plots showing the execution time and energy used to execute a hybrid program for all possible configurations, program SP (216 configurations[3]) on a Xeon cluster and program CP (400 configurations[4]) on an ARM cluster respectively. Each configuration in these plots is a tuple consisting of the number of nodes, number of cores and the

[3] $n \in [1, 2, 4, 8, .., 256], c \in [1..8]$ and $f \in [1.2, 1.5, 1.8]GHz$
[4] $n \in [1, 2, 3, .., 20], c \in [1..4]$ and $f \in [0.2, 0.5, 0.8, 1.1, 1.4]GHz$

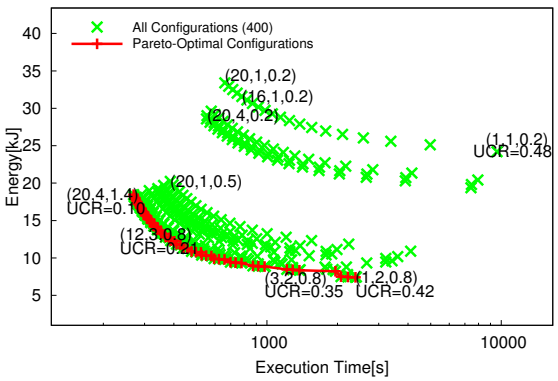

Figure 9: ARM cluster executing CP program

core clock frequency (n, c, f). For each configuration point, the x-axis denotes the program's execution time and the y-axis represents the corresponding energy used. 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*.

These time-energy plots illustrate three counter-intuitive insights. Firstly, as the execution-time deadline is relaxed, the configurations have lesser number of nodes but surprisingly use lesser energy. Decreasing the number of nodes decreases power used but increases execution time, and thus it is expected that the energy ($power \times time$) will be constant. Although, decreasing the number of nodes causes a linear decrease in the power used, the effect on the execution time is non-linear and is characterized by the queuing delays due to network contention ($T_{w,net}$, $T_{s,net}$). Secondly, as the energy budget is reduced, counter-to-intuition, the number of cores and core clock frequency increases. Increasing the number of cores or core clock frequency reduces execution time but increases power. Although the increase in the power is a factor of processor design, the decrease in execution time is not linear and is characterized by shared-memory contention ($T_{w,mem}$, $T_{s,mem}$). Thirdly, Pareto-optimal configurations do not necessarily use all available cores operating at the maximum frequency, e.g. ARM system configuration (3,2,0.8) is on the Pareto frontier of the CP program.

B. Useful Computation Ratio (UCR)

While Computation-to-Communication Ratio (CCR) is a widely used metric to quantify the communication costs (both inter and intra-node) of a parallel program and a higher CCR implies better efficiency, this metric is not normalized and hence is less useful for making comparisons across configurations. As HPC applications become increasingly data-centric and with the widening gap between floating-point speed and memory bandwidth [18], it is very important to characterize the performance of a program with respect to an upper bound to compare and evaluate its execution across a large system configuration space. To address this, we propose a new metric called the Useful Computation Ratio (UCR) of a hybrid program as:

$$UCR = \frac{T_{useful}(= T_{CPU})}{T} \tag{13}$$

Total execution time, T, for a hybrid program is defined as:

$$T = T_{CPU} + T_{data\_dep} + T_{mem\_contention} + T_{net\_contention} \tag{14}$$

Since T_{CPU} is defined as the time spent by the program in the system for useful computations including overlapped data accesses (Equation 2), the maximum value for normalized UCR is one. T_{data_dep} is a program characteristic, and does not change for a given program with a fixed input size executing on a specific system architecture. As computations need data to be fetched, $T_{mem\_contention}$ represents the communication cost of fetching data from shared-memory within a node and $T_{net\_contention}$ accounts for the inter-node communication cost. Thus, UCR is a useful measure for comparing the execution efficiencies of a hybrid program across different system configurations.

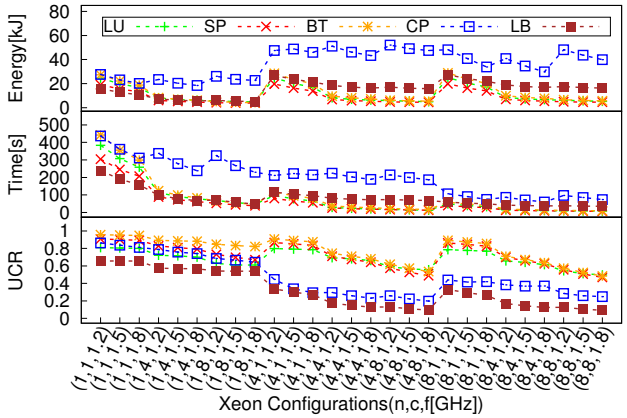

Figure 10: UCR and time-energy performance on Xeon cluster

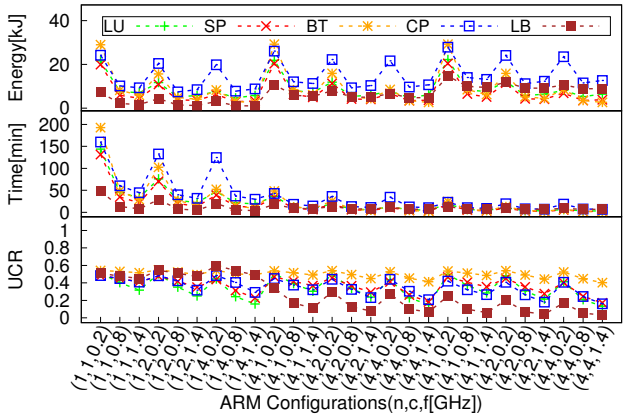

Figure 11: UCR and time-energy performance on ARM cluster

Figures 10 and 11 plot UCR and the time-energy performance of five hybrid programs for different configurations. For a given program and an input size, the upper bound of UCR is obtained for an execution configuration with a single node, single core and the lowest operating frequency, $(1, 1, f_{min})$, as this configuration incurs negligible communication overheads.

The differences between the CISC and RISC ISA of the processors causes UCR for Xeon to be much higher (0.96 for BT program) than UCR for ARM (0.54 for BT program).

UCR not only exhibits resource mismatches between computational processing and communication resources of a system but also expresses mismatches in the program implementation due to an imbalance in the parallelism among logical processes versus parallel threads. The CP and LB programs illustrate this imbalance, as seen from the steep drop in the UCR values (Figures 10 and 11) with increasing number of logical processes and threads. Increasing the number of nodes, or cores or core clock frequencies, increases contention for shared resource and thus decreases UCR. However, increasing the number of nodes, or cores or core clock frequencies, reduces both the execution time and energy used for certain configurations. Hence, while the UCR metric gives useful information regarding the balance between the computation and communication resources in a system for a program, it cannot be used to determine efficient execution configurations, as configurations with high UCR are not necessarily energy-efficient.

Figures 8 and 9 also show the UCR values for some Pareto-optimal configurations. An increase in the execution time results in lesser number of nodes in the Pareto-optimal configurations. This decrease in the number of nodes, reduces contention thus increasing the UCR as observed. While the Pareto-optimal configurations are energy efficient, they do not necessarily imply a high UCR. As is observed from Figures 8 and 9, the UCR values of the Pareto-optimal configurations (0.05 to 0.67 for Xeon and 0.10 to 0.42 for ARM) is quite small compared to the best possible UCR (0.91 for Xeon and 0.48 for ARM). Hence there is room for further optimizing the Pareto configurations for better balance between the computation and communication phases of the hybrid parallel program on a specific system architecture.

Optimizing UCR for Pareto-optimal configurations: UCR represents the balance between the execution rates and the communication rates of resources in a system and hence can be improved by either changing the program or system design to achieve a better matching between these rates. For example, doubling the memory bandwidth reduces the number of stall cycles due to shared-memory contention by two times, and thus improves the UCR of SP program executed on Xeon configuration (1,8,1.8) from 0.67 to 0.81. This increase in UCR also reduces the execution time by 7 seconds and energy used by 590 Joules, thus further optimizing the Pareto-frontier configuration. Hence, the proposed approach can be easily applied by system architects to gain insights into resource imbalances, and further optimize the Pareto-frontier using UCR. Secondly, for a given system configuration, application developers can fine-tune their implementations by re-structuring the iterations during the computational and communication phases of the program for different l (= n) and τ (= c) to further improve the UCR of Pareto-optimal configurations. The proposed approach thus offers a holistic hardware-software co-design by gaining useful insights from the predicted execution time, energy and UCR of hybrid parallel programs.

VI. Conclusions

While hybrid parallel programs offer the dual-advantage of scalability via distributed-memory and better performance using shared-memory, users of these programs face an uphill task in determining the time-energy optimal set of logical processes (ℓ) and threads (τ) for executing the program. From a system's perspective, this challenge translates to determining energy efficient execution configurations in terms of (n, c, f), where n is the number of nodes, c the number of cores and f is the operating core clock frequency. This paper presents an approach to determine time-energy Pareto-optimal system configurations (n, c, f) for executing a hybrid program using a measurement-driven analytical model. The proposed model addresses the effects of using both distributed-memory and shared-memory communication by considering inter and intra-node resource overlaps, memory contention among cores within a node and network contention across multiple nodes.

We validate the proposed approach for a range of HPC programs from different domains such as non-linear partial differential equation solvers, electronic structure calculations and computational simulation for fluid dynamics. These representative HPC applications are validated against direct measurement on Intel Xeon and ARM Cortex-A9 clusters as they have a diverse time-energy performance. Validation results show a mean error of less than 15% between the predicted and measured execution time and energy. We show that a Pareto frontier consisting of time-energy Pareto-optimal configurations exist for a hybrid program executed on a homogeneous cluster. 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, users of hybrid programs can easily apply our approach for time-energy efficient execution. To further optimize the Pareto frontier, we introduce a new metric, useful computation ratio (UCR) that quantifies the degree of resource contentions and communication overheads in an execution. We also show how system architects and application developers can increase the UCR of Pareto-optimal configurations by balancing resource service demands with resource utilization, to further minimize system inefficiencies.

References

[1] OpenLB: http://www.openlb.net, *[online]*.

[2] WattsUpMeters: https://www.wattsupmeters.com/secure/index.php

[3] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent, HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs, *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[4] AMD, AMD to Accelerate the ARM Server Ecosystem with the First ARM-based CPU and development Platform from a Server Processor Vendor: http://www.amd.com/en-us/press-releases/Pages/amd-to-accelerate-2014jan28.aspx, *[online]*, Jan. 2014.

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

[6] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, R. Thakur, Fine-grained Multithreading Support for Hybrid Threaded MPI Programming, *International Journal of High Performance Computing Applications*, 24(1):49–57, 2010.

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

[8] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, S. Reinhardt, The M5 Simulator: Modeling Networked Systems, *MICRO*, 26(4):52–60,2006.

[9] S. Cho, R. G. Melhem, On the Interplay of Parallelization, Program Performance, and Energy Consumption, *Trans. on Parallel and Distributed Systems*, 21(3):342–353, 2010.

[10] J. Choi, M. Dukhan, X. Liu, R. Vuduc, Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks, *Proc. of 28th IPDPS*, 2014.

[11] J. W. Choi, D. Bedard, R. Fowler, R. Vuduc, A Roofline Model of Energy, *Proc. of 27th IPDPS*, pages 661–672, May 2013.

[12] I.-H. Chung, S. R. Seelam, B. Mohr, J. Labarta, Tools for Scalable Performance Analysis on Petascale Systems, *Proc. of IPDPS*, 2009.

[13] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, D. S. Nikolopoulos, Online Power-performance Adaptation of Multithreaded Programs using Hardware Event-based Prediction, *Proc. of 20th ICS*, 2006.

[14] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, M. Schulz, Prediction models for Multi-dimensional Power-performance Optimization on many Cores, *Proc. of 17th PACT*, pages 250–259, 2008.

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

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

[17] J. Demmel, A. Gearhart, B. Lipshitz, O. Schwartz, Perfect Strong Scaling using no Additional Energy, *Proc. of 27th IPDPS*, 2013.

[18] J. Dongarra, High Performance Computing - Future Directions, *Keynote of 43rd ICPP*, 2014.

[19] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, D. Burger, Dark Silicon and the End of Multicore Scaling, *Proc. of 38th ISCA*, pages 365–376, 2011.

[20] R. Ge, X. Feng, K. W. Cameron, Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters, *Proc. of SC*, 2005.

[21] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, R. M. Wentzcovitch, QUANTUM ESPRESSO: a Modular and Open-source Software Project for Quantum Simulations of Materials, *Journal of Physics: Condensed Matter*, 21(39):395502 (19pp), 2009.

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

[23] V. Heuveline, M. J. Krause, J. Latt, Towards a Hybrid Parallelization of Lattice Boltzmann Methods, *Computers Mathematics with Applications*, 58(5):1071–1080, 2009.

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

[25] C. H. Hsu, W. C. Feng, A Power-aware Run-time System for High-performance Computing, *Proc. of SC*, 2005.

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

[27] N. Kappiah, V. W. Freeh, D. K. Lowenthal, Just in Time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI programs, *Proc. of SC*, 2005.

[28] B. C. Lee, 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.

[29] D. Li, B. de Supinski, M. Schulz, D. Nikolopoulos, K. Cameron, Strategies for Energy-Efficient Resource Management of Hybrid Programming Models, *TPDS*, 24(1):144–157, Jan 2013.

[30] T. Li, D. Baumberger, D. A. Koufaty, S. Hahn, Efficient Operating System Scheduling for Performance-asymmetric Multi-core Architectures, *Proc. of SC*, 2007.

[31] M. Luo, X. Lu, K. Hamidouche, K. Kandalla, D. K. Panda, Initial Study of Multi-endpoint Runtime for MPI+OpenMP Hybrid Programming Model on Multi-core Systems, *Proc. of 19th PPoPP*, pages 395–396, 2014.

[32] C. Ma, Y. M. Teo, V. March, N. Xiong, I. R. Pop, Y. X. He, S. See, An Approach for Matching Communication Patterns in Parallel Applications, *Proc. of IPDPS*, 2009.

[33] G. Mao, D. Böhme, M.-A. Hermanns, M. Geimer, D. Lorenz, F. Wolf, Catching Idlers with Ease: A Lightweight Wait-State Profiler for MPI Programs, *Proc. of the 21th European MPI Users' Group Meeting*, 2014.

[34] D. S. McFarlin, C. Tucker, C. Zilles, Discerning the Dominant Out-of-order Performance Advantage: Is It Speculation or Dynamism?, *Proc. of 18th ASPLOS*, pages 241–252, 2013.

[35] M. K. Mehmet-Ali, J. F. Hayes, A. Elhakeem, Traffic Analysis of a Local area Network with a Star Topology, *Trans. on Communications*, 36(6):703–712, 1988.

[36] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, R. Rajkumar, Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling, *Proc. of 16th ICS*, pages 35–44, 2002.

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

[38] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, S. Vishin, Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era, *Proc. of DAC*, pages 174:1–174:9, 2013.

[39] R. Rabenseifner, G. Hager, G. Jost, Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-core SMP Nodes, *Proc. of 17th PDP*, pages 427–436, 2009.

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

[41] M. Sadiku, S. Musa, Local Area Networks, *Performance Analysis of Computer Networks*, pages 167–195. Springer International Publishing, 2013.

[42] V. Sarah, Dell offers 64-bit ARM Microserver Proof-of-concept for Hyperscale on the Heels of Open Compute Summit Momentum: http://en.community.dell.com/dell-blogs/dell4enterprise/b/dell4enterprise/archive/2014/02/04/dell-offers-64-bit-arm-microserver-proof-of-concept-for-hyperscale-on-the-heels-of-open-compute-summit-momentum, *[online]*, Feb. 2014.

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

[44] J. V.-N. Steven, Canonical claim the First ARM 64-bit Server production Software Deployment, Linux and Open Source: http://www.linuxtoday.com/upload/applied-micro-canonical-claim-the-first-arm-64-bit-server-production-software-deployment-140529135505.html, *[online]*, May. 2014.

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

[46] B. M. Tudor, Y. M. Teo, A Practical Approach for Performance Analysis of Shared-memory Programs, *Proc. of IPDPS*, pages 652–663, 2011.

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

[48] D. Turner, A. Oline, X. Chen, T. Benjegerdes, Integrating new capabilities into netpipe, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 37–44. Springer, 2003.

[49] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, J. Emer, Scheduling Heterogeneous Multi-cores through Performance Impact Estimation (PIE), *Proc. of ISCA*, pages 213–224, 2012.

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

[51] J. S. Vetter, M. O. McCracken, Statistical Scalability Analysis of Communication Operations in Distributed Applications, *Proc. of 8th PPoPP*, pages 123–132, 2001.

[52] J. S. Vetter, F. Mueller, Communication characteristics of large-scale scientific applications for contemporary cluster architectures, *Journal of Parallel and Distributed Computing*, 63(9):853–865, 2003.

[53] S. Williams, A. Waterman, D. Patterson, Roofline: an Insightful Visual Performance Model for Multicore Architectures, *Communications of the ACM*, 52(4):65–76, 2009.

[54] D. Wong, M. Annavaram, KnightShift: Scaling the Energy Proportionality Wall through Server-Level Heterogeneity, *Proc. of MICRO*, pages 119–130, 2012.

[55] D. H. Woo, H.-H. S. Lee, Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era., *IEEE computer*, 41(12):24–31, 2008.