Consolidating Applications for Energy Efficiency in Heterogeneous Computing Systems

Jing Zhang*, Hao Wang*, Heshan Lin*, and Wu-chun Feng*†
*Department of Computer Science
†Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia 24060
Email: {zjing14, hwang121, hlin2, wfeng}@vt.edu

Abstract

By scheduling multiple applications with *complementary* resource requirements on a smaller number of compute nodes, we aim to improve performance, resource utilization, energy consumption, and energy efficiency simultaneously. In addition to our naïve consolidation approach, which already achieves the aforementioned goals, we propose a new energy efficiency-aware (EEA) scheduling policy and compare its performance with current state-of-the-art policies, namely round-robin (RR), resource utilization-aware (RUA), adaptive shortest-job first (ASJF) in order to support the consolidation of applications in heterogeneous computing systems, and in turn, simultaneously improve performance, resource utilization, energy consumption, and energy efficiency, as measured by the energy-delay product. Of particular note, our experimental results on a real heterogeneous computing system demonstrate the efficacy of our scheduling policies by *improving overall energy efficiency by an order of magnitude*.

***Keywords*-consolidation, CPU, GPU, energy efficiency, scheduling, heterogeneous computing, resource utilization, benchmarks, energy-delay product**

I. Introduction

In high-performace computing (HPC), we have witnessed a dramatic increase of data-parallel architectures, such as graphics processing units (GPUs), due to their superior performance and energy efficiency compared to traditional CPUs. In the Top500 list [18] published in June 2013, 8.6% of the systems, including two of the top ten supercomputers, were equipped with GPUs. GPU-accelerated systems also accounted for three of the top five supercomputers in the latest Green500 list [2]. Despite the great potential of these GPU-accelerated heterogeneous computing systems, applications have difficulty in fully utilizing both the CPU and GPU. As a consequence, resource utilization is relatively low in these heterogeneous systems, as reported in recent studies [14].

One way to improve resource utilization of HPC systems is to consolidate applications, i.e., by scheduling multiple applications with complementary resource requirements on the same set of compute nodes. Although such consolidation has been extensively studied on CPU-based systems, there are unique challenges and opportunities in consolidating applications on heterogeneous systems. In particular, with the introduction of portable programming models such as OpenCL [8], an application can run on either CPUs or GPUs with the same source code. The challenge is to decide where to schedule an application kernel, i.e., on the CPU or GPU. Existing studies in consolidating applications via CPU-GPU co-scheduling have focused on performance [12], [13].

With power and energy consumption becoming first-order constraints in designing next-generation supercomputers [3], we investigate the feasibility of consolidating applications to reduce power and energy consumption while maintaining or improving performance in heterogeneous computing systems. In contrast to the performance-oriented consolidation of applications, which has been extensively studied [4], [12], [13], we seek to consolidate applications to reduce energy consumption and reap its associated benefits. Though the power consumption of modern GPUs, e.g., AMD Radeon HD 7970 at 250 W, may be quite high, energy consumption is the product of execution time and power. If the improvement in execution time on the GPU cannot offset its higher power consumption, scheduling an application on the GPU may lead to better performance but worse energy efficiency.

In this paper, our application workloads have source-code portability, i.e., they can run on either CPUs or GPUs with the same source code. Our motivation is to consolidate a set of such applications running on different nodes to one node having GPU and to schedule each of them running on CPU or GPU for better energy efficiency.

To evaluate our approach, we use the SNU NPB suite [1], which is a well-tuned NPB benchmark suite that uses OpenCL, as the driving applications. We first implement a naïve consolidation mechanism to schedule original CPU workloads running on CPU and GPU workloads running on GPU of the same node at the same time. The results have demonstrated the naïve consolidation approach can save 15% and 33% energy on two different machines, each with a GPU, respectively, with a mere 2% and 1% performance degradation due to contention on the CPU. The results also illustrate that the naïve consolidation mechanism cannot fully utilize the CPU and GPU resources.

We then propose an energy efficiency-aware (EEA) consolidation algorithm for a heterogeneous CPU+GPU system. Our algorithm adopts the ratio of the energy-delay product (EDP) on the CPU and GPU as the metric and first schedules the most CPU energy-efficient jobs to run on the CPU and most GPU energy-efficient jobs to run on the GPU. We compare our design with several scheduling algorithms in the recent literature, including round-robin (RR), resource utilization-aware (RUA) and adaptive shortest-job first (ASJF).

Our research makes the following contributions;

1) An evaluation of the performance and energy consumption of heterogeneous CPU+GPU computing systems — with and without consolidating applications. (The empirical results show that better energy consumption can be achieved even when using a naïve consolidation mechanism. Our analysis also indicates the opportunities to improve consolidation.)
2) An energy efficiency-aware (EEA) algorithm for consolidating application jobs to run on the CPU and GPU of the same node. (We evaluate our EEA algorithm against several well-known scheduling algorithms to demonstrate that our design can provide better performance and energy consumption.)

The rest of the paper is organized as follows. In Section II, we provide the necessary background about the topics discussed in this paper and distinguish our work from existing research in this area. In Section III, we first implement and evaluate a naïve consolidation strategy to demonstrate the benefit of consolidation and analyze the drawbacks of this naïve design. We then present our approach for energy efficiency-aware (EEA) consolidation. Section IV presents an evaluation of our energy efficiency-aware (EEA) consolidation algorithm and compare its efficacy with respect to the current state of the art. In Section V, we conclude the paper and introduce our future work.

II. Background and Related Work

We first provide a brief description about graphical processing units (GPUs) and the driving applications as background information, followed by a discussion of related work.

A. Programming on GPGPUs

Originally, a GPU was a special-purpose processor that was designed solely for graphics rendering. With the vertex and fragment shaders added to the graphics pipeline, GPUs are increasingly being used for general-purpose computation, and in turn, supporting general-purpose computation on GPUs (GPGPUs). With the delivery of programming models such as NVIDIA's Compute Unified Device Architecture (CUDA) [11], and more broadly, OpenCL [17], many applications have been parallelized on GPUs to take advantage of their computational horsepower.

Generally, a GPU contains a set of single-instruction, multiple-data (SIMD) streaming multiprocessors (SMs). Each SM consists of a set of scalar cores, as illustrated in Figure 1. On each SM, the on-chip memory, including register, shared memory, constant cache, and texture cache, can be accessed by threads executing on the SM. Shared by all SMs, the off-chip memory or global memory can be accessed by all threads on the GPU.

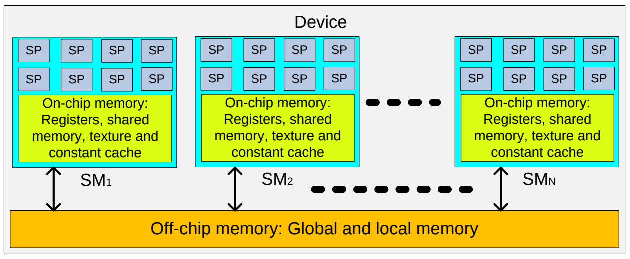

Device
SP
On-chip memory: Registers, shared memory, texture and constant cache
SM1
SM2
SMN
Off-chip memory: Global and local memory

Figure 1. An Overview of GPU Architecture

CUDA and OpenCL are arguably the most widely used programming models for the GPU. Unlike CUDA, OpenCL has the capability to execute on different hardware architectures. With OpenCL, programmers can write computational kernels once and execute them on various platforms such as the GPU or CPU. As a result, we choose OpenCL as the programming model in this paper to investigate the consolidation of applications on a heterogeneous CPU+GPU system.

B. The Driving Benchmarks

The NAS Parallel Benchmark (NPB) [10] suite is a widely used benchmark suite to evaluate the performance of parallel computing systems. The benchmarks in NPB are derived from computational fluid dynamics (CFD) applications. There are five kernels, including CG, EP, FT, IS, and MG, to simulate typical numerical methods in CFD applications and three pseudo-applications, including BT, LU, and SP, to simulate both computation and data access in CFD applications.

The SNU NPB suite is an OpenCL realization of NPB for heterogeneous parallel computing. It has been optimized for multicore CPUs and different generations of GPUs. Previous research [15] has illustrated that the performance of the OpenCL-based SNU NPB benchmark suite is comparable to or better than the performance of the original OpenMP implementations on multicore CPUs.

C. Related Work

Consolidation is a well-known technique for improving resource utilization. Over the last few years, significant research effort has been dedicated to the enhancement of consolidation algorithms. Hermenier et al. [6] proposed a consolidation manager called *Entropy*, which takes both local and global optimization into account when mapping tasks to nodes. Srikantaiah et al. [16] model consolidation as a modified bin-packing problem and reveal the energy-performance tradeoffs for consolidation. Chen et al. [4] construct a tool that is based on queuing theory to predict application scalability and to suggest consolidation decisions. However, the above research targets homogenous (CPU-only) computing systems.

For a heterogeneous computing system with a CPU and GPU, consolidation algorithms must consider additional challenges, such as heavy context-switching between different processes using the GPU, the data communication overhead between CPU and GPU, and so on. rCUDA [5] allows an application to use a CUDA-compatible device in a remote node. GPU utilization is improved by scheduling computational kernels from multiple applications on the same GPU. Context funneling [19] is proposed to avoid context-switching when multiple processes share a same GPU. By maintaining a shared context between multiple threads, the cost of heavy context-switches is avoided. GPU workload consolidation [9], [12] seeks to consolidate multiple workloads running on the same GPU. By automatically intercepting CUDA driver calls and analyzing thread-block utilization, the consolidation presented in [9] can gather multiple GPU workloads and issue different GPU kernels simultaneously to fully utilize GPU SMs. The consolidation framework presented in [12] provides the concept of an affinity score to allow more fine-grained control of GPU kernel consolidation. Furthermore, it does not require intercepting calls to the CUDA driver API. However, all the above work only targets GPU workloads while we seek to consolidate applications running on both CPU and GPU.

Concurrent application scheduling for CPU+GPU systems has investigated the performance of multiple scheduling algorithms on both single and multiple CPU+GPU nodes [13]. The scheduling algorithms in this paper only consider execution time as the optimization objective. In contrast, we also study energy efficiency, as measured by the energy-delay product (EDP). The energy-delay product explicitly refers to the energy consumption and execution time (i.e., delay), respectively, of an application. In turn, our proposed energy efficiency-aware (EEA) scheduling algorithm seeks to optimize for both.

III. Design of the Consolidation Algorithms

In this section, we first evaluate a naïve consolidation algorithm for consolidating applications on a heterogeneous CPU+GPU system in order to demonstrate the efficacy of consolidation in such systems. We then analyze and expose the shortcomings of the naïve consolidation algorithm. and in turn, propose a new algorithm for energy efficiency-aware (EEA) consolidation.

A. Naïve Consolidation Algorithm

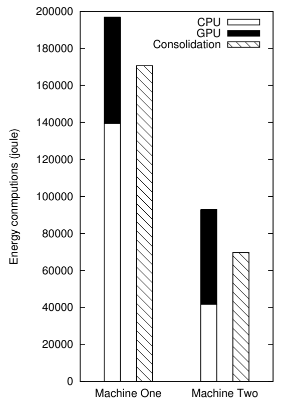
To naïvely consolidate applications, we simply schedule applications running on two different nodes to run on a single node by scheduling a CPU-based application on the CPU and a GPU-based application on the GPU. We present this case as a demonstration of the efficacy of consolidating applications on heterogeneous CPU+GPU systems.

Using the machines described in Table I as our evaluation platforms, Figure 2 shows the energy consumption and the execution time of applications with and without the naïve consolidation. “CPU” represents running all eight benchmarks of SNU NPB one by one on the CPU of a particular machine. “GPU” represents running the same benchmarks, one after another, on the GPU of a particular machine. “Consolidation” represents running the same benchmarks on the CPU and on the GPU of the same machine at the same time.

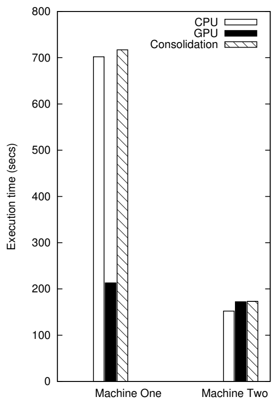
Figure 2(a) shows that the overall energy consumption when using naïve consolidation (i.e., running the SNU NPB becnhmarks on the CPU and GPU of the *same* node simultaneously) results in noticeably lower energy consumption than when running the SNU NPB benchmarks on the CPU and GPU of *different* nodes simultaneously. For machine one and machine two, the overall energy savings

	Machine One	Machine Two
CPU	Intel Xeon E5405 (dual quad-core), 6MB LLC	Intel Xeon E5-2665 (dual oct-core), 2*16 threads, 20MB LLC
GPU	AMD Radeon HD 7970 (ATI Tahiti)	NVIDIA Geforce GTX Titan (Nvidia Kepler)
Runtime Library	AMD APP SDK v2.8	NVIDIA CUDA 5.0
Test Set	SNU-NPB Class B	SNU-NPB Class B

Table I. Experimental Setup



(a) Energy Consumption



(b) Execution Time

Figure 2. Energy Consumption and Performance Comparison between CPU node, GPU node and Consolidation

is 15% and 33%, respectively. The reason for the energy savings is that with consolidation, we can eliminate the standby energy consumption of the second node by simply consolidating all the workload onto a single node.

Relative to performance, Figure 2(b) shows that with consolidation, there is marginal performance loss: 2% on machine one and 1% on machine two. The performance loss comes from contention on the CPU. Even though the CPU offloads the GPU workload to the GPU, the CPU still needs to orchestrate data movement into or out of the GPU as well as launch the execution of different computational kernels during the execution of the GPU workload.

Figure 3 plots the power consumption of machine one and machine two while running the SNU NPB benchmarks using the CPU only, GPU only, or both ("Consolidation"), as was similarly done for Figure 2. Looking at the results for machine one in Figure 3(a), we observe that after 280 seconds, the power consumption of the "Consolidation" approach drops significantly because the workloads running on GPU have finished. When the workloads running on the CPU finish after 710 seconds, the overall power consumption drops again to that of system standby. So, from 280 seconds to 710 seconds, only the CPU is executing any workload while the GPU is idle. Thus, while the naïve consolidation of applications does save energy, as shown in Figure 2, the results in Figure 3 illustrate that energy savings is not maximized as one resource is left idle (i.e., GPU) and consumes extra standby energy for 430 seconds.

In Figure 3(b) for machine two, we observe that the power consumption of "GPU" is not always higher than that of "CPU." This indicates that in heterogeneous computing systems that the GPU has the potential to not only provide better performance but also power consumption. As a consequence, the collective results from Figure 3 encourage us to design an energy efficiency-aware consolidation algorithm to optimize for reduced energy consumption.

B. Algorithm for Energy Efficiency-Aware (EEA) Consolidation

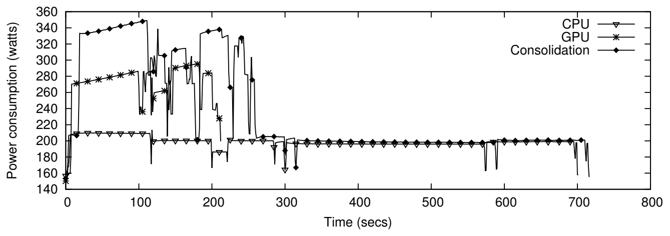
As energy efficiency depends on both execution time and energy consumption, the metric "Gflops/watt" is often used to compare the energy efficiency of different individual applications (or computational kernels) [7]. However, one could argue that "Gflops/watt" is not appropriate when scheduling a set of disparate applications, some of which may have operations other than floating-point operations (flop). An alternative metric to use is the *energy-delay product*, a commonly-used metric from circuit design. which is defined simply as the product of energy consumed and execution time. The larger the energy-delay product (EDP), the less energy efficient it is.

Algorithm 1 provides the details of our approach to energy efficiency-aware job scheduling, one that consolidates applications onto fewer nodes. We first calculate the ratio of the energy-delay product on the CPU to that on the GPU for each job. At Line 6, based on the ratio, we sort the job list to put the most energy-efficient CPU job at the head of the list and put the most energy-efficient GPU job at the tail of the list. From Line 10 to Line 17, if the CPU is idle, we schedule the most energy-efficient CPU job to CPU and the most energy-efficient GPU job to the GPU.

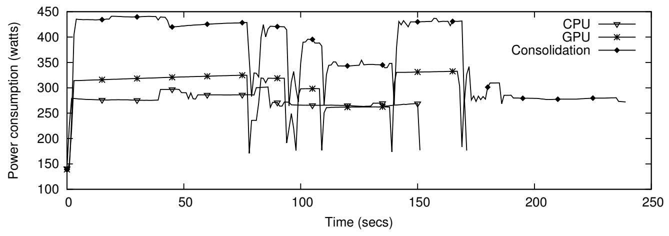
The benefits of our algorithm are two-fold. First, our algorithm keeps both the CPU and GPU busy with workloads running, thus improving resource utilization. Second, our algorithm greedily schedules the most energy-efficient CPU job to CPU and most energy-efficient GPU job to GPU so that the most energy savings can be obtained.

IV. Performance Evaluation

In this section, we first evaluate the execution time and the energy consumption of each benchmark on a CPU and and then on a GPU. We then evaluate state-of-the-art consolidation algorithms and compare them to



(a) Machine One



(b) Machine Two

Figure 3. Power Consumption of CPU, GPU, and CPU+GPU Consolidation on Two Machines

Algorithm 1 Energy Efficiency-Aware Consolidation

```
1: Input: $edp\_cpu$: the energy-delay product of application on CPU; $edp\_gpu$: the energy-delay product of application on GPU; $jobs$ : job list to be scheduled
2: procedure ENERGY EFFICIENCY-AWARE CONSOLIDATION($edp\_cpu, edp\_gpu, jobs$)
3:     for all jobs do
4:         $edp\_r_i = edp\_cpu_i \div edp\_gpu_i$
5:     end for
6:     sort($jobs$, $edp\_r_i$, ascending)
7:     $i \leftarrow 0$
8:     $j \leftarrow n - 1$
9:     while $i \mathrel{!=} j$ do
10:        if CPU is idle then
11:            schedule $job_i$ on CPU
12:            $i \leftarrow i + 1$
13:        end if
14:        if GPU is idle then
15:            schedule $job_j$ on GPU
16:            $j \leftarrow j - 1$
17:        end if
18:    end while
19: end procedure
```

our proposed algorithm for energy efficiency-aware (EEA) consolidation.

As introduced in Table I of Section III, we use two different machines, each populated with a GPU, for our heterogeneous computing systems. The first machine consists of two Intel Xeon E5405 quad-core CPUs and 4GB of RAM; the GPU is an AMD Radeon HD 7970. The second machine consists of two Intel Xeon E5-2665 oct-core CPUs and 16GB RAM; the GPU is a NVIDIA GTX Titan. As noted in Section II, we adopt SNU NPB suite as the benchmark suite. Because the problem size of Class C cannot fit into global memory of the GPU for the MG, FT, and BT benchmarks, we unify the problem size to Class B for all benchmarks.

A. Evaluation of the NPB Suite

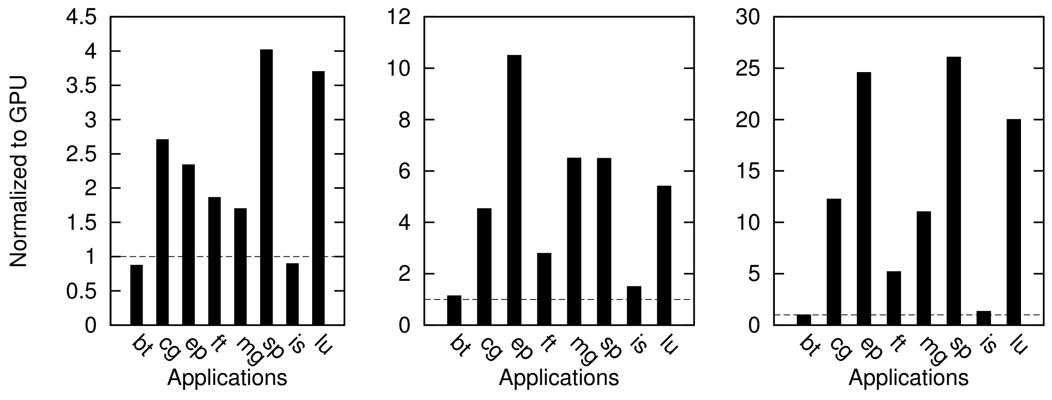
Figures 4 and 5 show the energy consumption, execution time, and energy-delay product (EDP) of running on a CPU relative to running on a GPU for machine one and machine two, respectively. For machine one, Figure 4(a) shows that the CPU consumes more energy than the GPU in all cases *except* BT and IS, where the CPU consumes less energy than the GPU. On the other hand, Figure 4(b) shows that all the application benchmarks take longer to run on the CPU than on the GPU, even BT and IS, which both consume less energy on the CPU. By looking at the energy-delay product (EDP) in Figure 4(c), we see that the product of the energy consumption and execution time shows that the GPU is more energy efficient for all the application benchmarks *except* BT, where the CPU is only very slightly more energy efficient.

For machine two, which consists of Intel Sandy Bridge quad-core CPUs (E5-2665) and a NVIDIA Geforce GTX Titan GPU, the CPU performs better than the GPU with respect to both execution time and energy efficiency for the BT, CG, FT, and IS benchmarks. One potential reason for this is that the GPU kernels in OpenCL NPB have been optimized for multiple GPU generations, especially for NVIDIA Fermi but *not* for the latest NVIDIA Titan GPU. Because the NVIDIA Titan GPU is based on the latest Kepler architecture, additional optimizations using new features of Kepler arechitecutre should be added in OpenCL NPB to more fully optimize and utilize the Kepler-based NVIDIA Titan GPU.

B. Evaluation of the EEA Consolidation Algorithm

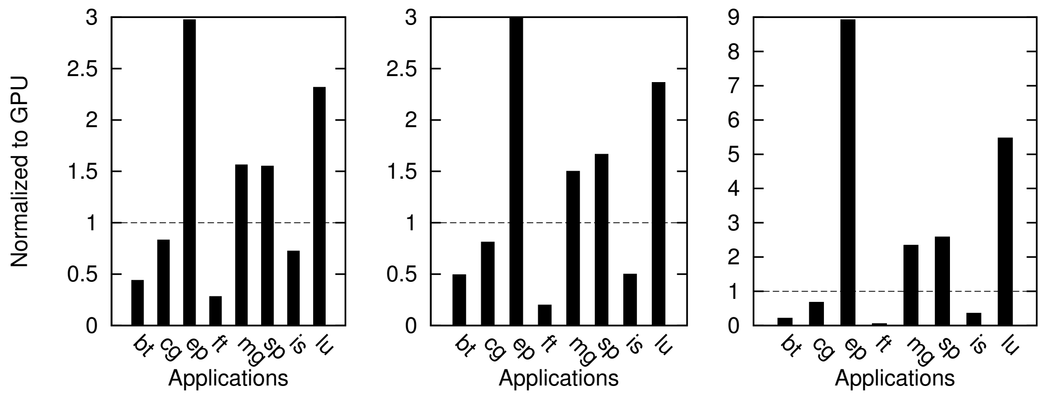
Here we evaluate and compare our energy efficiency-aware (EEA) consolidation algorithm to several well-known scheduling algorithms for consolidation, as noted below.

- *Round Robin (RR)* is one of the simplest scheduling algorithms. It schedules same number of jobs to CPU



(a) Energy Consumption (b) Execution Time (c) Energy-Delay Product

Figure 4. Machine One: Energy Consumption, Execution Time, and Energy-Delay Product of the CPU Relative to the GPU



(a) Energy Consumption (b) Execution Time (c) Energy-Delay Product

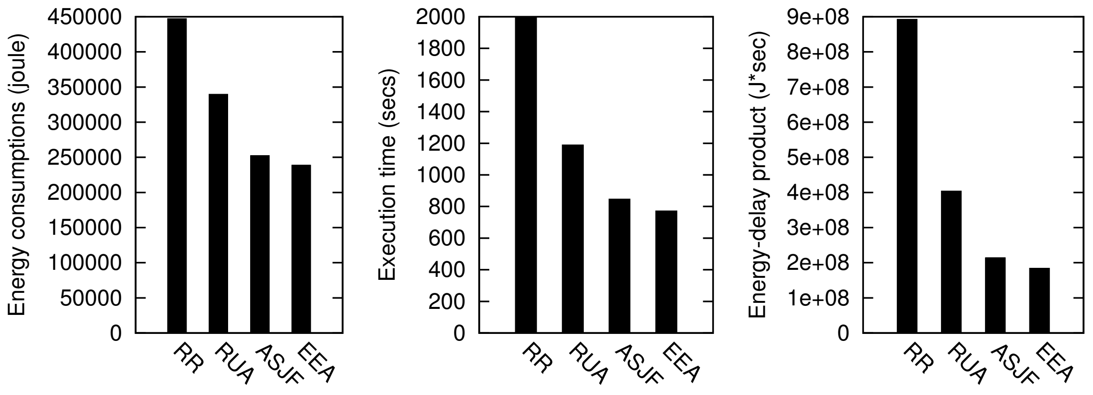
Figure 5. Machine Two: Energy Consumption, Execution Time, and Energy-Delay Product of the CPU Relative to the GPU

and GPU without any priority consideration. Since it does not guarantee that applications running on CPU have similar execution time to those running on GPU, the RR-based consolidation would likely result in low utilization of computing resources.

- *Resource Utilization-Aware (RUA)* is designed to improve resource utilization through scheduling a job to run on the current idle device. Compared to RR, RUA can improve resource utilization, but it cannot provide good performance since it treats the CPU and GPU as equivalent devices in this scenario.
- *Adaptive Shortest-Job First (ASJF)* is based on the shortest-job-first algorithm. By selecting a shortest job to be running on a currently available device (CPU or GPU), this algorithm tries to guarantee improved resource utilization and improved performance at the same time.

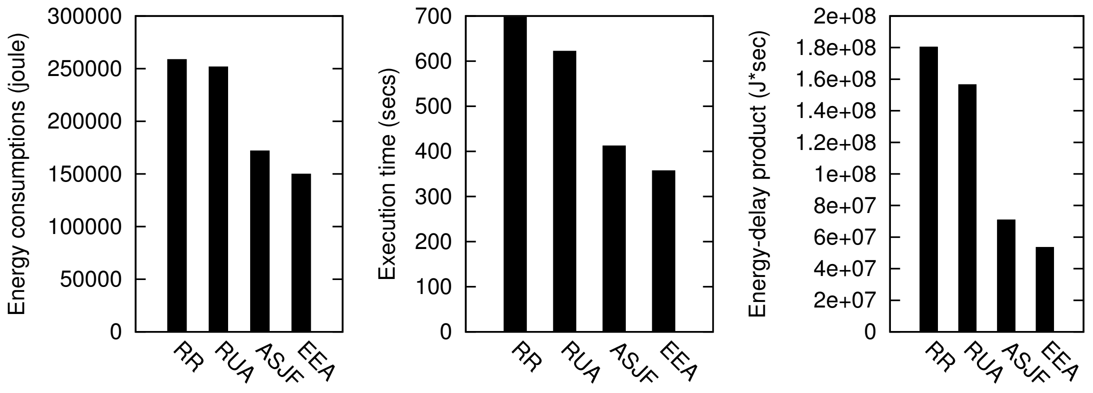
In our experiments, we randomly generate forty jobs, consisting of benchmarks from SNU NPB suite. We then use the collected execution time and energy consumption of each benchmark on the CPU and on the GPU as parameters for ASJF-based consolidation and our energy efficiency-aware (EEA) consolidation.

Figures 6 and 7 chart the energy consumption, execution time, and energy-delay product of different consolidation algorithms on our two heterogeneous test platforms, machine one and machine two, respectively. The figures show that our EEA consolidation algorithm performs the best across all three evaluation metrics: (1) lowest energy consumption, (2) fastest execution time, and (3) lowest energy-delay product $\rightarrow$ best energy efficiency. On machine one, EEA improves energy consumption by 87%, 42%, and 5%; execution time by 158%, 54%, and 9%; and energy efficiency by 384%, 119%, and 16% when compared to RR, RUA, and ASJF, respectively. On machine two, EEA



(a) Energy Consumption (b) Execution Time (c) Energy-Delay Product

Figure 6. Machine One: Energy Consumption, Execution Time, and Energy-Delay Product for Round-Robin (RR) Consolidation, Resource Utilization-Aware (RUA) Consolidation (RUA), Adaptive SJF (ASJF) Consolidation, and Energy Efficiency-Aware (EEA) Consolidation



(a) Energy Consumption (b) Execution Time (c) Energy-Delay Product

Figure 7. Machine Two: Energy Consumption, Execution Time, and Energy-Delay Product for Round-Robin (RR) Consolidation, Resource Utilization-Aware (RUA) Consolidation (RUA), Adaptive SJF (ASJF) Consolidation, and Energy Efficiency-Aware (EEA) Consolidation

improves energy consumption by 72%, 68%, and 14%; execution time by 95%, 74%, and 15%; and energy efficiency by 237%, 192%, 32% when compared to RR, RUA, and ASJF, respectively. Thus, these figures illustrate that our energy efficiency-aware (EEA) consolidation algorithm can improve both energy consumption and execution time, and in turn, the energy-delay product.

V. Conclusion and Future Work

In this paper, we target the problem of consolidating applications on heterogeneous systems with GPUs. We demonstrate that even naïve consolidation, which schedules original CPU workloads to the CPU and GPU workloads to the GPU onto a single node, can improve energy consumption, but that it is far from an optimal approach. Thus, we propose an energy efficiency-aware (EEA) consolidation algorithm to improve the performance and energy consumption of a heterogeneous CPU+GPU system. Our experiments show that the EEA algorithm can significantly improve energy consumption, execution time, and energy-delay product when compared to other consolidation approaches that are based on well-known scheduling algorithms, including round-robin (RR), resource utilization-aware (RUA), and adaptive shortest-job first (ASJF), on two generations of compute nodes. As part of our future work, we will extend our energy efficiency-aware (EEA) consolidation algorithm to multiple nodes with multiple GPUs and integrate it into open-source job schedulers.

Acknowledgment

This work is in part supported by NSF via CNS-0916719.

References

[1] SNU NPB Suite. http://aces.snu.ac.kr/Center_for_Manycore_Programming/SNU_NPB_Suite.html.

[2] The Green 500, Ranking the World's Most Energy-Efficient Supercomputers. http://www.green500.org/.

[3] S. Amarasinghe, D. Campbell, W. Carlson, A. Chien, W. Dally, E. Elnohazy, M. Hall, R. Harrison, W. Harrod, K. Hill, J. Hiller, S. Karp, C. Koelbel, D. Koester, P. Kogge, J. Levesque, D. Reed, V. Sarkar, R. Schreiber, M. Richards, A. Scarpelli, J. Shalf, A. Snavely, and T. Sterling. ExaScale Software Study: Software Challenges in Extreme Scale Systems. http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[4] L. Y. Chen, D. Ansaloni, E. Smirni, and W. B. A. Yokokawa. Achieving Application-Centric Performance Targets via Consolidation on Multicores: Myth or Reality? In *Proceedings of the 21st ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*, 2012.

[5] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti. Performance of CUDA Virtualized Remote GPUs in High Performance Clsuters. In *Proceedings of the International Conference on Parallel Processing (ICPP'11)*, 2011.

[6] F. Hermenier, X. Lorca, J. M. Menaud, G. Muller, and J. Lawall. Entropy: A Consolidation Manager for Clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'09)*, 2009.

[7] C.-h. Hsu, W.-c. Feng, and J. S. Archuleta.

[8] Khronos OpenCL Working Group. The OpenCL Specification (version 1.1), June 2010. http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf.

[9] D. Li, S. Byna, and S. Chakradhar. Energy-Aware Workload Consolidation on GPU. In *Proceedings of the 2011 International Conference on Parallel Processing Workshops (ICPPW'11)*, 2011.

[10] NASA Advanced Supercomputing Division. NAS Parallel Benchmarks. http://www.nas.nasa.gov/publications/npb.html.

[11] NVIDIA. Compute Unified Device Architecture. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[12] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar. Supporting GPU Sharing in Cloud Environments with A Transparent Runtime Consolidation Framework. In *Proceedings of the 20th ACM International Symposium on High-Performance Paralle and Distributed Computing (HPDC'11)*, 2011.

[13] V. T. Ravi, M. Becchi, W. Jiang, G. Agrawal, and S. Chakradhar. Scheduling Concurrent Applications on A Cluster of CPU-GPU Nodes. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*, 2012.

[14] T. K. Samuel, S. McNally, and J. Wynkoop. An Analysis of GPU Utilization Trends on the Keeneland Initial Delivery System. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment(XSEDE'12)*, 2012.

[15] S. Seo, G. Jo, and J. Lee. Performance Characterization of the NAS Parallel Benchmarks in OpenCL. In *Proceedings of the 2013 IEEE International Symposium on Workload Characterization (IISWC'11)*, 2011.

[16] S. Srikantaiah, A. Kansal, and F. Zhao. Energy Aware Consolidation for Cloud Computing. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems (HotPower'08)*, 2008.

[17] Stone, J.E. and Gohara, D. and Guochun Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science Engineering*, 12(3):66–73, 2010.

[18] TOP500 Supercomputing Sites. TOP500 Supercomputing List. http://www.top500.org/.

[19] L. Wang, M. Huang, and T. El-Ghazawi. Towards Efficient GPU Sharing on Multicore Processors. *ACM SIGMETRICS Performance Evaluation Review*, 40(2):119–124, 2012.