Power-Aware Dynamic Task Scheduling for Heterogeneous Accelerated Clusters

Tomoaki Hamano, Toshio Endo and Satoshi Matsuoka
Tokyo Institute of Technology/JST, Japan

Abstract

Recent accelerators such as GPUs achieve better cost-performance and watt-performance ratio, while the range of their application is more limited than general CPUs. Thus heterogeneous clusters and supercomputers equipped both with accelerators and general CPUs are becoming popular, such as LANL's Roadrunner and our own TSUBAME supercomputer. Under the assumption that many applications will run both on CPUs and accelerators but with varying speed and power consumption characteristics, we propose a task scheduling scheme that optimize overall energy consumption of the system. We model task scheduling in terms of the scheduling makespan and energy to be consumed for each scheduling decision. We define acceleration factor *to normalize the effect of acceleration per each task. The proposed scheme attempts to improve energy efficiency by effectively adjusting the schedule based on the acceleration factor. Although in the paper we adopted the popular EDP (Energy-Delay Product) as the optimization metric, our scheme is agnostic on the optimization function. Simulation studies on various sets of tasks with mixed acceleration factors, the overall makespan closely matched the theoretical optimal, while the energy consumption was reduced up to 13.8%.*

1 Introduction

Acceleration in supercomputing and clusters is recently receiving considerable attention as the next primary methodology for achieving multi-petaflops and beyond, as well as for conserving space and energy from desktops to large supercomputers. Already, the LANL's Roadrunner [10] has reached the 1 Petaflop pleateau using the IBM/Sony/Toshiba Cell processor in the June 2008 Top500 [14]. Our own TSUBAME supercomputer has increased its performance by a factor of two over the last 2 years using acceleration via a sophisticated heterogeneous Linpack algorithm [5] utilizing the ClearSpeed accelerator [4], and more recently, with the addition of NVIDIA Tesla GPUs with extensions to the algorithm, with only a minor increase in the overall power consumption.

Such accelerators usually consist of highly multi-threaded and/or SIMD computing processing elements that are highly concentrated on a single silicone die. In order to control highly parallel threads and vector compute engines, their programming models are fairly restricted, usually SIMD/SPMD in nature, focusing on vector/stream processing oriented applications. Moreover, they usually do not run their own OS themselves, but rather rely on conventional processors to run the core OS and conduct non compute-bound tasks such as I/O. This is in contrast to general-purpose multi-core processors whose level of integration is smaller, with only several compute units on a single die, but whose applicability is quite broad with their superior single-thread performance and flexibility. Moreover, in current mainstream architectures, some accelerators may reside on the I/O bus, further compromising their execution speed due to frequent data transfers between the CPU and the accelerator.

Given such a fairly complex heterogeneity that exists even within a single compute node, for a particular application, it is not always given that accelerators will be superior to conventional CPUs in terms of the processing speed as well as energy consumption. Some applications may achieve an order of magnitude or more performance boost, but in other cases, they may be even slower than a CPU. A more subtle situation arises as portability of code is realized between CPUs and accelerators. For example, the Brook+ [1] language runs on both AMD GPUs and CPUs, and NVIDIA has announced that upcoming version of CUDA [13] will support executions of CUDA programs on multi-core PCs. Moerover, OpenCL [9] is also expected as the basis for heterogeneous computing. Another situation would be that a library with same specifications and interfaces might exist for both CPUs and accelerators. In a clus-

ter/supercomputing environment when there would be multitudes of CPU and accelerator resources available, possibly not only an intra-node but also an inter-node heterogeneous fashion—on TSUBAME 1.2., 318 nodes have 2 NVIDIA Tesla 10p GPUs plus a Clearspeed accelerator, whereas 330 have only Clearspeeds but no Teslas, and 90 nodes are CPU only—it is not obvious how one would schedule incoming stream of jobs to optimize system throughput and overall energy consumption.

In other words, for a single application, existing energy-aware technology, plus performance prediction or profiling would be largely sufficient to optimize its runtime and/or energy, but a simple collection of such optimizations might not result in optimality for either or both metrics. For example, for a machine consisting of 10 CPUs and 10 GPUs, a set of 10 jobs might arrive that would run 10% faster on a GPU over CPU, and consume the approximately the same energy. A simple scheduler would then schedule the jobs on the GPUs. Then, another set of 10 jobs that would run 10 times faster on a GPU and thus would consume approximately an order of magnitude less energy could arrive. A simple scheduler would have no choice but to schedule the jobs inefficiently on CPUs, but a smarter scheduler might simply queue the jobs if energy consumption is a significant factor. Alternatively, given a profile of jobs, the scheduler might have set the threshold for GPU scheduling higher so that the first set of jobs would have been scheduled on CPUs rather than GPUs in the first place.

We propose a task scheduling that optimize overall energy consumption for heterogeneous clusters that consist of CPUs and accelerators. There are static and dynamic version of the scheme, the latter being more favorable in practice and demonstrated to be as almost as effective as the static version. We model task scheduling in terms of the scheduling makespan and energy to be consumed for each scheduling decision. For simplicity we assume that a given task can be executed on both CPUs and accelerators—a situation where a task would only run on one or the other can easily be modeled by extending the makespan to infinity. We define the *acceleration factor* to normalize the effect of acceleration for each task. The proposed scheme attempts to improve the energy efficiency by effectively making scheduling decision based on various parameters and the state of scheduling including the acceleration factor as well as the results of its own scheduling. Although in the paper we adopted the popular EDP (Energy-Delay Product) as the optimization metric, our scheme is agnostic on the optimization function.

Simulation studies on various sets of tasks with mixed acceleration factors, the overall makespan closely matched the theoretical optimal, while the energy consumption was reduced up to 13.8% compared to the ECT (Earliest Completion Time) heuristics.

2 Related Work

There has been numerous work on applying CPU DVFS (Differential Voltage and Frequency Scaling) feature control for power-aware scheduling, minimizing the overall parameters such as Energy or EDP [3, 11]. For example, Chen, et al. propose a task scheduling scheme for a set of tasks described as a DAG, and propose to save energy by executing the tasks that are not on the critical paths with lower voltage and frequency. Here, they assume the processors to be homogeneous, but the problem to be heterogeneous. Moreover, the degree of heterogeneity afforded between CPUs vs. accelerators could be one or two orders of magnitude, which is not the case for DVFS. In fact we do not employ DVFS in our work, but such a feature can be orthogonally added. As for heterogeneity, there has historically been much work in the area stemming back to the 1970s, where most focused on makespan only and theoretically derived worst- or average case characteristics. A new breed of work has been conducted recently on heterogeneous scheduling on grid environments, such as the work by Casanova [2], where the effect of file I/O is considered in a heterogeneous environment in the proposed heuristics. Another work by Topcuoglu, et al. [15] considers a DAG application and compared two scheduling schemes HEFt and CPOP. Again, both focus on makespan only, and moreover, does not directly address when there could be order(s) of magnitude difference in Performance/Watt.

3 Modeling Scheduling in a Heterogeneous Accelerated Computing Environment

First, we present an overview of our scheduling scheme, and model the task execution time (makespan), total system energy consumption, and energy efficiency.

3.1 Overview of our Task Scheduling Scheme

Firstly, for simplicity we assume that each compute node consists of a single CPU and arbitrary numbers of accelerators per node. Also, we assume that the CPUs and accelerators are homogeneous to one another, i.e.,

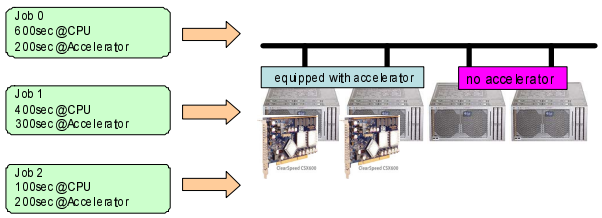

Figure 1. Overview of an Accelerated and Heterogeneous Cluster

heterogeneity within the node is that between the CPU and GPU, whereas inter-node heterogeneity is in terms of number of GPUs per node(Figure 1). Although more complex models can be built, such variations can be encoded in terms of the mentioned heterogeneity.

Secondly, we assume that a given program can be executed on both CPUs and accelerators. By all means acceleration factor will differ on a task-by-task basis, as is with unrelated environments. We also assume that each task is independent and sequential, i.e., there are no dependencies between the tasks, they do not span across multiple nodes nor migrate or be subdivided. In practice, accelerators might consume some fraction of CPU power due to handling of host-accelerator communication [5], but we do not consider such effects for now.

3.2 Modeling Task Scheduling in an Accelerated and Heterogeneous Environment

3.2.1 Task Execution Time

As mentioned earlier, we define a parameter *Acceleration Factor* in order to normalize the effect of acceleration over execution on a CPU, as another parameter *task size*. The latter is the execution time of a task on a CPU, whereas the former is the ratio of execution of the same task on a CPU and the accelerator, and bigger this value is the effect of acceleration is greater. More formally, given a task size t_{cpu}C and acceleration factor α, the time required for the task to execute on an accelerator (t_{acc}) is given by:

$$t_{acc} = t_{cpu}/\alpha \tag{1}$$

Tasks that can only be executed on a CPU or an accelerator can be modeled by setting t_{acc} or t_{cpu} to infinity, respectively.

3.2.2 Energy Consumption of Heterogeneous Tasks

Next, in order to estimate energy consumption, we further introduce two parameters P_{busy} and P_{idle}, which expresses the power consumption for busy and idle periods, respectively. We assume that P_{busy} is constant across tasks, and P_{idle} for CPUs is standby power when the CPU is idle without any ongoing computation, while P_{idle} for accelerator is the delta increase in power consumption on a node when one accelerator is added, and that accelerator is not being utilized.

Given such parameters, the overall energy consumption of each compute node ($E_{processor}$) can be given as follows:

$$E_{processor} = busyTime \times P_{busy} + (systemTime - busyTime) \times P_{idle} \tag{2}$$

$busyTime$ indicates the sum of overall execution time on a particular node, and $systemTime$ is the total execution time of the entire cluster system, i.e., $systemTime = max\{busyTime\}$. Using these, we can estimate the overall energy consumption of the entire system (E_{all}) to be as follows:

$$E_{all} = \sum_{processor} E_{processor} \tag{3}$$

3.2.3 Energy Efficiency Metrics

One metric that has been used in low power modeling of existing process is generalized Energy-Delay Product given by $ED^n(n \geq 1)$ [7]. Here, the value of n provides flexible emphasis on the tradeoffs between execution time and energy consumption. Here, we employ the most widely used EDP where $n = 1$ where emphases are both linear:

$$EDP = E_{all} \times systemTime \tag{4}$$

4 Our Proposed Hetero / Acceleration Aware Scheduling

Here we give the details our proposed heterogeneous / acceleration task scheduling algorithm that aims to optimize the EDP metric. The basic strategy is to attempt to schedule the tasks with highest acceleration factor onto the accelerators to shorten the makespan and thereby optimize the energy usage. The problem is how to do so with dynamic, non a-priori scheduling.

One simple heuristics to achieve this is to set a certain threshold on the acceleration factor (e.g., $\alpha > 1$),

and all tasks that are above this threshold be scheduled on the accelerator, and otherwise on the CPUs. Although this would be fine on an individual task level, it may not be optimal at the system level in terms of throughput. As an extreme example, consider the case where the acceleration factors of all tasks are either $\alpha > 1$ or $\alpha < 1$. In such a case, either the CPU or the accelerator resource will remain idle, which is not preferable both in terms of system throughput and energy usage due to idle power consumption. Rather, we must consider the global property of the set of tasks and make scheduling decisions, while still being dynamic in the decision making.

4.1 Deciding CPU / Accelerator Scheduling

We now present our scheme to make effective dynamic decisions on CPU / Accelerator scheduling. The basic scheme is to conduct virtualized scheduling on a virtualized set of resources, and find optimal tradeoff point in the EDP function.

More concretely, we first define virtual CPUs and virtual accelerators as follows. When there are n CPUs in the system, virtualized CPUs can execute a task with tasksize t_{cpu} in time t_{cpu}/n, and power during execution and idle being nP_{busy},nP_{idle}, respectively. Virtual accelerators are defined in a similar fashion. Next, We put all tasks for virtual CPUs and compute the EDP. We then repeat the following: we pick the task with the largest acceleration factor from the ones assigned to the virtual CPUs, assign it to a virtual accelerator, and recomputed the EDP. If there are multiple tasks with the same acceleration factor, we pick the largest one. Repetitions of this sequence will yield a series of EDP values, out of which we pick the assignments with the minimum EDP.

Although this scheme can be effective, its utility will depend on which set of tasks we consider as known a-priori. The next section will evaluate the scheme under following two assumptions regarding this issue:

- We assume that information regarding all the tasks are known beforehand, and schedule the tasks statically as described above (proposed).
- When we schedule a task, we make dynamic decisions based only on the information we have scheduled up to the point of that task (dyn-proposed). Although we must re-compute the task assignments as described above for each task as above, we can optimize this by memorizing the past scheduling results; in scheduling the kth task, the compute complexity of scheduling would be $O(\log k)$. Although this could still be a problem for a very large k for a long-running system, we could further optimize this by only memorizing the last K tasks which would give the finite bound.

5 Evaluation

Given the scheme above, we evaluate the scheme under the assumption that we would have abundant nodes with CPU/GPU configuration as described, such as our TSUBAME 1.2 supercomputer, and that there would be both a CPU-only version as well as a CPU-GPU accelerated version, under a simulated environment.

In order to precisely evaluate the effectiveness, we first compared our scheme with existing simple scheduling schemes, and then with an optimal scheduling result computed by directly solving the linear programming problem. As existing schemes, we employ the standard ECT(Earliest Completion Time) [8] scheduling, and also threshold-1 scheduling, where for the latter any task with acceleration factor greater than 1 would execute on the accelerator, and otherwise on the CPU—this is equivalent to assigning each job to the fastest resource for each particular task.

5.1 Simulation Environment

As accelerators, we assume a GPU-accelerated environment such as our TSUBAME 1.2 supercomputer. The power consumption and the associated parameters are as seen in Table 1. These numbers were taken from earlier prototype configurations of TSUBAME's accelerated nodes, where there are 4 NVIDIA 8800GTS accelerator boards, hosted by a single AMD Phenom 9850 quad-core CPU, and executed optimized GotoBLAS [6] (representing compute-intensive tasks) and NukadaFFT [12] (representing bandwidth-intensive tasks) routines varying the number of GPUs.

As for task sets, we prepared several scenarios with respect to the mixture in their acceleration factors, as seen in Table 2. This allows us to observe the effect of the task mix on our scheduling schemes under various foreseeable circumstances in a large system with significant number of jobs. For task_set4, we assumed that we have three sets of tasks, those that are not executable on GPUs, BLAS, and FFT, and their mix being 20%,40%,40%, respectively. Each task set embodies 10,000 tasks, The task priority was decided randomly. The size of the tasks were determined so that it would take 1 to 1000 seconds when executed on a CPU. We generate the four workloads at a time and then reuse in all tests except the comparison to optimal scheduling. Finally we assume that there are 50 CPUs coupled

Table 1. power

Comp. Resource	Exec. Power	Idle Power
CPU	220(W)	130(W)
Accelerator(GPU)	100(W)	50(W)

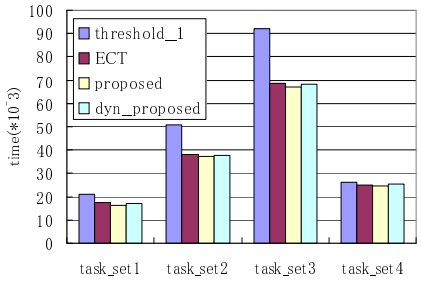

Figure 2. Scheduling Makespan

with 50 accelerators.

5.2 Evaluation Results

5.2.1 Scheduling Makespan

Figure 2 shows the makespans for different task sets, comparing existing scheduling schemes versus our proposed schemes. We observe that, compared to ECT, we obtain maximum 7.5% improvement over ECT and maximum 27.2% improvement over threashold-1 scheduling. In fact, we achieve improvements for all scenarios and task sets compared to ECT, which might be counter-intuitive. The reason for our superiority is that, when we need to schedule tasks with high acceleration factors in a successive fashion, when all the accelerators are busy, ECT will go ahead and schedule the task onto CPUs, whereas with our algorithm, it might make a decision to schedule the task onto an accelerator, yielding a better overall result. To confirm this, we deliberately sorted the priorities for task_set2 in the descending order of its acceleration factors; the result was quite disastrous for ECT, where it experienced 39% slowdown.

We further observed that, task sets with larger numbers of higher acceleration factors tend to perform better with our scheme. Also, our dynamic scheduling scheme (dyn-proposed) worked almost as good as our static scheme, performing favorably versus threshold-1 as well as ECT scheduling.

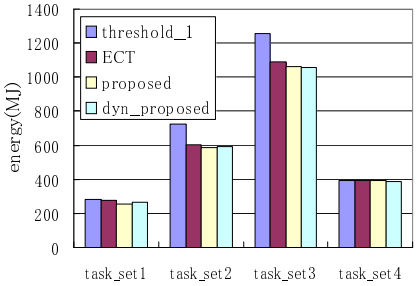

Figure 3. Energy Consumption of Different Scheduling Schemes

5.2.2 Energy Consumption

Figure 3 shows the energy consumption numbers of different scheduling schemes. We observe that, despite higher power consumption of accelerators, our scheme achieves up to 7.5% reduction in energy consumption over ECT, and likewise 18.7% over threshold-1. The reason is as follows: our scheme effectively identifies the fact that tasks with higher acceleration factors can be executed with lower energy values, and assigns them properly to accelerators. On the other hand, the energy savings was not as dramatic as makespan reduction in comparison with threshold-1 scheduling. This is because modern GPU accelerator idle power consumption being very efficient, and as a result, leaving them idle will drastically reduce the overall energy consumption, and as such, lengthening of the makespan has smaller effect compared to CPUs where subsystems esp. memory continue to consume large amounts of energy despite the CPU being idle.

5.2.3 Comparing the EDP

Figure 4 shows the improvements of EDP of each task set, normalized to that of threshold-1 scheduling. For all task sets we observe improvements, ranging from 5.5% up to 41%. We note that the improvement is smaller for task_set1 and task_set4, both of which embody larger number of tasks with higher acceleration factors. This is largely due to the reason that, since the number of accelerators are comparable to the number of CPUs (50), even with threshold-1 scheduling where most of the tasks would be assigned to accelerators and CPUs would remain idle, the overall processing went rather fast, so the makespan would still be short and the effect of CPU idleness had minor impact. Figure 5 shows the case when we reduce the number of accelerators to 10; in this case, the ineffective utilization

Table 2. task_set

Accel. Factor	task_set1	task_set2	task_set3	task_set4
0	0%	0%	0%	20%
0.0625-0.125	0%	12.5%	35.5%	0%
0.125-0.25	0.5%	12.5%	30%	0%
0.25-0.5	3%	12.5%	15%	0%
0.5-1	6%	12.5%	10%	0%
1-2	10%	12.5%	6%	0%
2-4	15%	12.5%	3%	40%(2.2)
4-8	30%	12.5%	0.5%	40%(5.15)
8-16	35.5%	12.5%	0%	0%

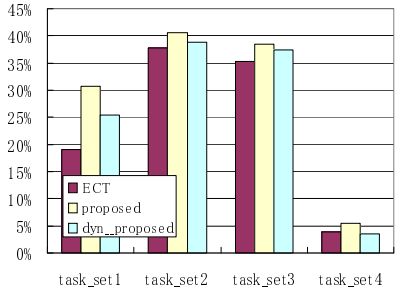

Figure 4. EDP Improvements with 50 Accelerators

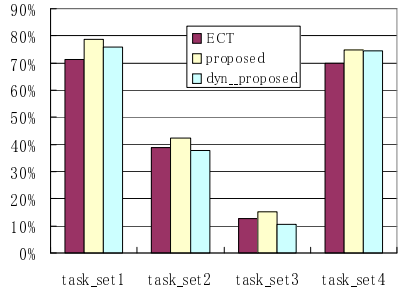

Figure 5. EDP Improvements with 10 Accelerators

of CPUs take effect, making EDP suffer this shortcoming. We also note that for all cases our scheme achieves higher EDP than ECT.

5.2.4 Comparing to Optimal Scheduling

Next, we created a scheduling oracle using linear programming with complete prior knowledge of the tasks and their acceleration factors, optimizing makespan or energy consumption for each case. The linear programming problem for makespan optimization can be formalized as follows:

$$
\begin{array}{ll}
minimize & systemTime \\
subject\ to & T_p(x) = \sum_{j=1}^{n} t_{cpu} x_{pj} (if\ p\ is\ CPU) \\
& T_p(x) = \sum_{j=1}^{n} t_{cpu}/\alpha\ x_{pj} (if\ p\ is\ Accelerator) \\
& T_p(x) \leq systemTime \\
& \sum_{p=1}^{m} x_{pj} = 1 \\
& x_{pj} = 0 or 1
\end{array}
$$

Similarly, energy minimization would be the following:

$$
\begin{array}{ll}
minimize & \sum_{p=1}^{m} \left((systemTime - T_p(x))\, P_{idle} + T_p(x) P_{busy} \right) \\
subject_to & T_p(x) = \sum_{j=1}^{n} t_{cpu} x_{pj} (if\ p\ is\ CPU) \\
& T_p(x) = \sum_{j=1}^{n} t_{cpu}/\alpha\ x_{pj} (if\ p\ is\ Accelerator) \\
& T_p(x) \leq systemTime \\
& \sum_{p=1}^{m} x_{pj} = 1 \\
& x_{pj} = 0 or 1
\end{array}
$$

Here, if $x_{pj} = 1$ then we would execute task j on processor p. Also, $T_p(x)$ denotes the execution time on processor p,and $systemTime$ would be $max_p T_p(x)$.The objective function for energy minimization P_{idle},P_{busy} would be that for CPUs and GPUs according to their architectural types. We also have additional constraints to properly model the system and various scheduling schemes

Given such a framework, we set the number of CPUs and accelerators to 10 each respectively, and the number of tasks to be 1000, to control the solution time for our LP solver, and compared it to the results of our scheduling schemes. Table 3 shows the makespan and energy consumption normalized to the optimal. We note that, even with the heuristics of our scheme, we

Table 3. Comparing Our Schemes to Optimal Scheduling

		task_set1	task_set2	task_set3	task_set4
proposed	makespan	1.030	1.026	1.048	1.024
	energy	1.014	1.025	1.029	1.013
dyn-proposed	makespan	1.129	1.056	1.064	1.038
	energy	1.088	1.019	1.023	1.075

obtain essentially optimal values for both irrespective of the task sets, confirming that our scheme is a good candidate for optimal dynamic scheduling for mixed CPU-accelerated environments of future clusters.

6 Conclusion and Future Work

We proposed a simple but effective scheduling scheme for mixed CPU and accelerator cluster environments, which we believe would be the dominant HPC architecture of the future, as exemplified by new classes of machines such as LANL's Roadrunner and our own TSUBAME supercomputer. In order to estimate the makespan and energy consumption, we model the task scheduling, and present a simple scheduling scheme based on the model. Our scheme, which can be either static or dynamic, presents a parameter called acceleration factor that represents how much each task can be accelerated if it would be executed on an accelerator rather than on a CPU. Our scheduling scheme would use the parameter as well as other system parameters to make effective scheduling decision to minimize the EDP of each task, which in turn would optimize both makespan and energy. Performance measurements shows that both static and dynamic versions of the scheme work very well, both in terms of makespan improvements as well as saving energy for various task sets with varying degree of acceleration factor mixes, achieving up to 13.8% improvement in makespan as well as up to 13.9% improvements in energy consumption compared to the standard ECT scheduling practiced in most centers today. As future work, we are considering the followings:

- Improving Model Accuracy for Heterogeneous Environments: The current model does not take into account the effect of CPU performance drainage required for accelerator support. For example, we observed up to 2.6 times slowdown for a CPU when 4 GPUs are idle versus when they are executing fully along with CPUs. A similar phenomenon is seen on the new NVIDIA Tesla Units on TSUBAME 1.2 running Linpack. Although the model is not definite, this is a combined effect of bandwidth consumption plus some servicing work required for the CPUs to support the GPUs. We are planning to come up with an effective model for such overhead so that it can be reflected properly in our model.

- Extensions to full on-line scheduling: Although our scheme has a dynamic version, it is still effectively offline scheduling. In order to be effectively useable in a standard batch queue system, we must adapt our model and algorithm to work as a standard online scheduling scheme.

References

[1] AMD. Stream Computing User Guide rev1.3.0, 2008.

[2] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *9th Heterogeneous Computing Workshop*, pages 349–364, 2000.

[3] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan. Reducing Power with Performance Constraints for Parallel Sparse Applications. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[4] ClearSpeed Inc. http://www.clearspeed.com/.

[5] T. Endo and S. Matsuoka. Massive Supercomputing Coping with Heterogeneity of Modern Accelerators. In *22th IEEE International Parallel and Distributed Processing Symposium (IPDPS'08)*, 2008.

[6] K. Goto. GotoBLAS. http://www.tacc.utexas.edu/resources/sofware/.

[7] C.-H. Hsu, W. chun Feng, and J. S. Archuleta. Towards Efficient Supercomputing:A Quest for the Right Metric. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[8] O. H. Ibarra and C. E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM (JACM)*, 24:280–289, 1977.

[9] Khronos Group. Open Computing Language. http://www.khronos.org/opencl/.

[10] Los Alamos National Laboratory. http://www.lanl.gov/orgs/hpc/roadrunner/.

[11] N. K. D. K. Lowenthal and V. W. Freeh. Just in Time Dynamic Voltage Scaling:Exploiting Inter-Node Slack to Save Energy in MPI Programs. In *ACM/IEEE Conference on Supercomputing (SC '05)*, 2005.

[12] A. Nukada, Y. Ogata, T. Endo, and S. Matsuoka. Bandwidth Intensive 3-D FFT kernel for GPUs using CUDA. In *IEEE/ACM International Conference on Supercomputing (SC08)*, 2008.

[13] NVIDIA Corp. CUDA Programming Environment. http://www.nvidia.com/.

[14] Top500 Supercomputers. http://www.top500.org/.

[15] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task Scheduling Algorithms for Heterogeneous Processors. In *8th Heterogeneous Computing Workshop*, pages 3–14, 1999.