Enabling GPU and Many-Core Systems in Heterogeneous HPC Environments using Memory Considerations

Francesc Guim
Computer Architecture Department
Technical University of Catalonia (UPC)
Intel Corporation
Barcelona, Spain
fguim@ac.upc.edu

Ivan Rodero
NSF Center for Autonomic Computing
Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
Piscataway, NJ, USA
irodero@cac.rutgers.edu

Julita Corbalan
Computer Architecture Department
Technical University of Catalonia (UPC)
Barcelona, Spain
juli@ac.upc.edu

Manish Parashar
NSF Center for Autonomic Computing
Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
Piscataway, NJ, USA
parashar@cac.rutgers.edu

***Abstract*—Increasing the utilization of many-core systems has been one of the forefront topics these last years. Although many-cores architectures were merely theoretical models few years ago, they have become an important part of the high performance computing market. The semiconductor industry has developed Graphical Processing Units (GPU) systems that provide access to many cores (i.e: Larrabee, Fermi or Tesla) that can be used for General Purpose (GP) computing.**

In this paper, we propose and evaluate a scheduling strategy for GPU and many-core architectures for HPC environments. Specifically, our strategy is a variant of the backfilling scheduling policy with resource sharing considerations. We propose a scheduling strategy that considers the differences between GP processors and GPU computing elements in terms of computational capacity and memory bandwidth. To do this, our approach uses a resource model that predicts how shared resources are used in both GP and GPU/many-core elements. Furthermore, it considers the differences between these elements in terms of performance. First, it models their differences in terms of computational power and how they share the access to the node's memory bandwidth. Second, it characterizes how the processes are allocated to the GPU.

Using this resource model, we design the Power Aware resource selection policy, which we combine with the Less-Consume scheduling policy. Our strategy tries to allocate jobs aiming at reducing the memory contention and the energy consumption. Results show that the scheduling strategies proposed in this work are able to save over 40% of energy and improve the system performance up to 30% with respect to traditional backfilling strategies.

I. Introduction

During this decade many computing architectures have been proposed by the High Performance Computing (HPC) community. Probably one of the most relevant topics that has been discussed over these previous years is the usage of multi/many core systems. These systems provide to applications access to many computing elements that can be used for threaded applications in order to do massive parallel processing in a single chip.

In the multi/many core systems research, one of the forefront topics is the usage of Graphical Processing Units (GPU) for General Purposes (GP) in scientific computing. Graphic applications (i.e.: ray tracing) require high parallelism, high data transfer and specific hardware units (i.e.: shaders or vector units). Although this hardware can be very specialized, GPUs can be also used to run non-graphic specific applications. Many recent research efforts have conducted studies about how to migrate scientific applications to GPU architectures. Furthermore, other studies have proposed several techniques to build HPC using GPUs ([1][2][3][4]).

Recently, many-core GPU systems oriented to HPC have been presented by the industry. Examples of these new systems are Larrabee [5], Fermi [6] or Tesla [7]. Although they were originally designed to run Graphic applications, they provide compatible instruction sets and APIs oriented to speed up the development of applications. This fact indicates that they probably will be good candidates to build new HPC architectures. However scheduling and resource selection strategies should be aware of these new architectures' characteristics to efficiently execute HPC workloads. Specifically, these policies should efficiently orchestrate the job allocations and the job scheduling considering the limitations of GPUs. Furthermore, they should efficiently manage this heterogeneous system composed by traditional GP processors and these many-core GPUs. For instance, they should consider the impact of co-allocating job processes in both GP and GPU cores on the system performance.

In [8], Hamano et al. proposed task scheduling techniques for heterogeneous clusters composed of accelerated systems

by GPU. The main idea of Hamno's work is to use Dynamic Voltage Scaling (DVS) techniques to synchronize tasks running in GP processors and tasks running in the GPUs. They assume that only one task is allocated per GPU. Thereby, using only one core per GPU, the proposed techniques do not exploit the massive level of parallelism that the GPU can provide. On the other hand, they do not consider how other resources (such as the memory bandwidth) may slowdown the tasks' performance.

The idea of our work is similar to the proposal of Hamano et al. but applied to job scheduling. We consider resource sharing contention, using all the available cores and co-allocation in GP and GPU cores. We also propose the usage of DVS to reduce the energy consumption when processes of a job do not need all the GP computing power. In these situations, the other processes of the same job can either experiment memory contention or be allocated to GPU cores. Different to Hamano et al., we try to exploit all the parallelism that GPUs can provide. In order to predict the memory bandwidth contention at the node and inside the GPU, we extend the resource usage model that we presented in [9].

In this paper, we consider both nodes composed of GP processors (such as Xeon architectures) and GPU/many-core (such as Larrabee) chips, and nodes composed only by GP processors. We propose a scheduling strategy that considers the differences between GP processors and GPU computing elements, in terms of computational capacity and memory bandwidth. To do this, our scheduling strategy uses a resource model that predicts how shared resources are used in both GP and GPU/many-core. Furthermore, it considers their differences in terms of performance. To do this, first, it models their differences in terms of computational power and how they share the access to the node's memory bandwidth. Second, it characterizes how the processes are allocated to the GPU. This is done modeling the socket (from the GPU to the node) memory bandwidth. This memory bandwidth is shared by all the GPU cores. Our performance studies reveal that the backfilling variants proposed in this paper can provide improvements of up to 30% in the bounded slowdown, and energy savings of up to 40% with respect to traditional backfilling strategies (e.g: EASY-Backfilling).

The rest of this paper is organized as follows. Section II presents the related work and the contributions of this paper. Section III describes the models used by our scheduling strategy. Section IV presents the scheduling strategy that we propose in this paper. Section V and VI describe the evaluation methodology that we have followed in our experiments, and the results obtained from simulations, respectively. Finally, section VII concludes the paper and outlines future research directions.

II. Related Work and Contributions

Authors like Feitelson, Schwiegelshohn, Calzarossa, Downey or Tsafrir have modeled logs collected from large scale parallel production systems. Their work provided inputs for the evaluation of different system behavior. Such studies have been fundamental to understand how the HPC centers' users behave and how the resources of such centers are used. Feitelson et al. analyzed logs from specific centers [10], provided approaches on general job and workload modeling [11][12][13], and, together with Tsafrir, provided approaches on detecting workload anomalies and flurries [14]. Calzarossa et al. [15][16], provided workload modelization surveys. Other authors such as Cirne et al. [17][18], Sevcik [19] and Downey [20] presented workloads models for moldable jobs. These studies have been considered in the design of new scheduling strategies.

From the early nineties, batch scheduling has been one of the main research topics in the area of HPC. Backfilling policies have been deployed in many HPC centers. The EASY-Backfilling, which was proposed by Lifka et al. [21], is the most simple backfilling policy. Many variants of EASY-Backfilling have been proposed in several approaches such as Shortest Job Backfilled First [22],[23] or LXWF-Backfilling [24]. General descriptions of the most frequently used backfilling variants and parallel scheduling policies can be found in the report that Feitelson et al. provide in [25].

The growing scales of HPC centers have made issues related to energy efficiency of critical concern. Power and cooling rates are increasing every year, and are becoming the dominant part of IT budgets. Therefore, in the last years, terms like "green scheduling" or "eco-scheduling" have become very popular. The main goal of these trends is to reduce the energy consumption in HPC systems while maintaining similar quality of service provided to users and resource utilization. Many approaches have addressed the usage of DVS techniques in the area of HPC systems [26][27]. In the area of processor architectures, Merkel et al. [28], present techniques that, considering thread memory contention, reduce cores frequency to balance its synchronization points. Thus, critical threads run at maximum frequency and the others run at lower frequency in order to finish at the same time. Although these techniques are multi-core oriented, they can be extrapolated to HPC environments having similar memory contention issues. In the work presented in this paper, we use a similar approach to balance the processes running in the GPU cores and the other processes running in the GP processors.

Other approaches have applied DVS techniques in task and job scheduling policies. Lawson et al. [29], propose a set of scheduling policies to reduce the power consumption in period of low loads. Hoon et al. [30], present a set of techniques to synchronize bag-of-tasks with deadlines by reducing the the CPU frequency. In the area of job

scheduling strategies different approaches and studies have been presented. For instance, Rofouei et al. [31], present job scheduling techniques to improve the processors lifetime based on power aware management policies. In our approach, we exploit all the different cores that are available in the system. We consider co-allocation in both GPU and GP cores. Furthermore, we also consider the usage of the memory bandwidth and its impact on the performance that the processes can achieve.

In [9], we evaluated the impact of considering the penalty introduced in the job runtime due to resource memory bandwidth sharing on the performance for backfilling policies. To achieve this, we developed a job scheduler simulator (Alvio simulator) that, in addition to traditional features, it implements a job runtime model and resource model that estimates the penalty introduced in the job runtime when sharing resources. Although in our previous work we only considered in the model the penalty introduced when sharing the memory bandwidth of a computational node, results showed a clear impact of our approach on system performance metrics such as the average bounded slowdown or the average wait time. Using the conclusions reached in our preliminary work, in [32] we described two new resource selection policies that are designed to minimize the saturation of shared resources. These interfaces were later used to design the Resource Usage Aware Backfilling [33].

In the work presented in this paper we extended our previous work in the following aspects:

- We have extended the original cluster model [9] in order to model clusters composed by heterogeneous computing resources.
- We have extended the processor model that we used in order to consider different levels of CPU frequency and voltage, include a energy consumption model and a new processor model (the GPU).
- We have extended the original runtime model in other to consider the memory bandwidth contention at two different levels: at the node level (all processors share the main memory bandwidth), and at the GPU socket level (all the GPU cores share a same access to the socket that is connected to the node).
- We have proposed an new resource selection policy that, considering the new resource and runtime models, tries to reduce the energy consumption. Also, we have proposed a scheduling strategy that uses this resource selection policy and the LessConsume policy [32] to improve the system performance and reduce the consumed energy.

III. The System Model

In this section, we provide a characterization of the resource model used by our proposed scheduling strategies. First, we briefly introduce the runtime model that we designed to evaluate the resource sharing in the Alvio simulator and that has been inherited in kento-perf. Second, we provide a description of how this runtime model has been used to characterize the HPC environment that the paper evaluates. In the last part of this section, we describe the DVS model that has been included in kento-sim in order to evaluate the energy consumption.

A. The runtime model

1) Modeling the memory bandwidth: In our simulator[1], the resource selection policy uses a reservation table. The reservation table represents the status of the system at a given moment and is linked to the architecture. The reservation table is a bi-dimensional table where the *X* axis represents the time and the *Y* axis represents the different processors and nodes of the architecture. It contains the running jobs allocated to the different processors during the time. In a simulation instance, two different reservation tables are used. One reservation table is used with estimated input values to model the scheduling prior the real execution (i.e.: estimated runtime, estimated memory, etc.), and the other one is used to model what is really going on in the system. This last one uses the real execution values (i.e.: real runtime, real memory bandwidth used, etc.) since models the real scheduling outcome.

Using the real resource requirements of all the allocated jobs, we compute the resource usage of the different resources available in the system. Thus, using the reservation table we are able to compute, at any point of time, the amount of resources that are being requested within each node. In this extended model, when a job α is allocated in the reservation table during the interval of time $[t_x, t_y]$ to processors $p_1, .., p_k$[2] that belong to nodes $n_1, .., n_j$, we check if any of the resources that belong to each node is overloaded during any part of the time interval. In the affirmative case, a runtime penalty will be added to the jobs that belong to the overloaded subintervals. To model these properties we defined the shared windows and the penalty function associated to them. A *shared window* is an interval of time $[t_x, t_y]$ associated to the node n where all the processors of the node satisfy the condition that either no process is allocated to the processor or the given interval is fully included in a process that is running in the processor. Intuitively, this condition defines an interval of time where the sharing resource demand is approximately constant.

To estimate the job runtime penalty due to memory bandwidth contention, first, the memory requested by all jobs allocated in the node is computed. Second, the penalty associated the memory bandwidth consumption is computed. This is a linear function that depends on the overload of the used resource. Thus, if the amount of memory bandwidth

[1]The runtime model presented in this subsection is deeply described and evaluated in [9].

[2]Henceforth, we will use the term *bucket* to refer to an interval of time in a given processor.

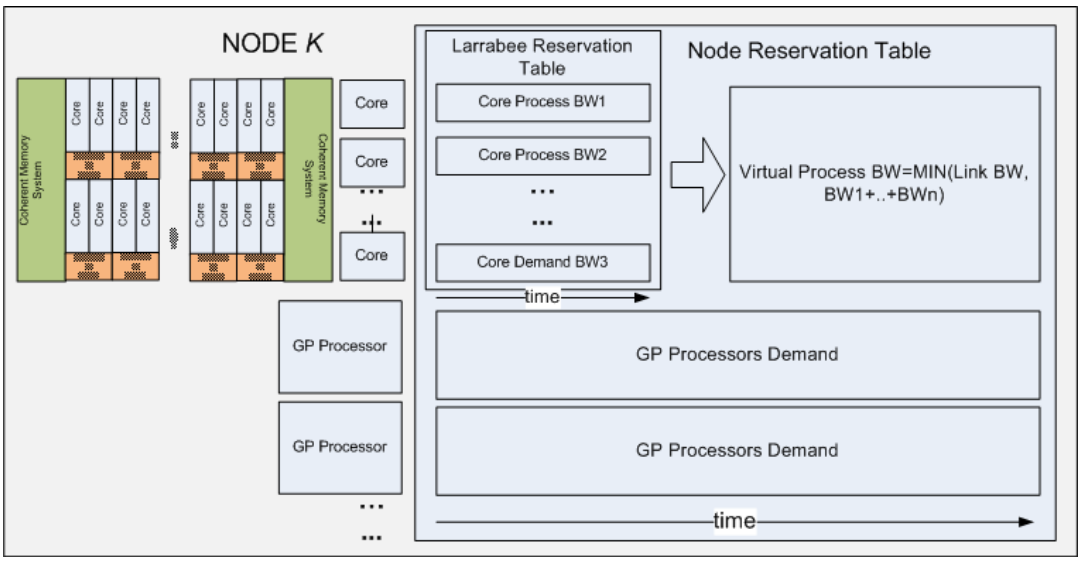


Figure 1: Reservation tables

is lower than the capacity, the penalty is zero. Otherwise, the penalty added is proportional to the fraction of demand and availability. Finally, the penalized time is computed by multiplying the length of the shared window and the penalty. This penalized time is the amount of time that will be added to all the jobs that belong to the window. This model has been designed for the memory bandwidth shared resource and is applicable to other shared resources with similar behavior. However, for other typology of shared resources, such as the network bandwidth, this model is not applicable.

To compute the penalized time that is finally associated to all the running jobs, first, the shared windows and the penalized times associated with each of them are computed for all the nodes. Second, the penalties of each job associated to each node are computed adding the penalties associated to all the windows where the job runtime is considered.

2) Modeling GPU and the system nodes: The model used to estimate the penalty due to memory bandwidth contention in the GPU socket and the node is the Reservation Table. Thereby, in the simulation model two different reservation tables are instantiated per node in the cluster model (see figure 1). At the GPU level, one reservation table is used to model how the processes that are running in the GPU cores are sharing the socket memory bandwidth. At the node level, one reservation table is used to model how all the processes running in GP processors and GPU are sharing the memory bandwidth of the node. This reservation table is an extended version of the reservation table that we presented in [9]. Like the original one, it models how the memory bandwidth is shared by all the jobs. However, a new type of element that encapsulates the GPU demand is introduced: the virtual processor.

In the reservation table associated to a node, a virtual process is permanently allocated in the virtual processor requesting the memory bandwidth that the GPU socket requests. The virtual process has been added to facilitate the computation of the job runtime penalties due to the memory bandwidth contention. A new virtual job is created when the bandwidth requested by the GPU socket changes (see figure 2). This occurs when the status of the GPU reservation table changes. With the presented abstraction of the GPU processes, the reservation model that we have described previously can be used to estimate the runtime penalties due to the saturation of the memory bandwidth of the node.

Figure 2 presents a possible snapshot of the status of a node's reservation table in a specific scheduling time stamp. Note that the reservation table, as described in our previous work, is updated each time that determined events occur (i.e: job termination, job runtime underestimation, job memory underestimation, job allocation, etc.). Thereby, the presented status may change as soon as any of the expected events occurs. In this example, processes of jobs *A*, *B* and *C* are allocated in the node. During this interval of time, there are three virtual processes allocated in the virtual GPU processor. The creation of *Virtual Process 2* is caused due to the GPU memory bandwidth demand has changed. On the one hand, this change can be caused due to a job that has a process allocated to one of the GPU cores has finished its execution. On the other hand, a new process may be allocated in the GPU and increased the memory demand. In this example three shared windows are defined. The first one includes *Virtual Process 1* and *Job B*. The second one includes the first part of both *Job A* and *Virtual Process 2*, and *Job C*. Finally, the last one includes the last part of *Virtual Process 2* and *Job A*.

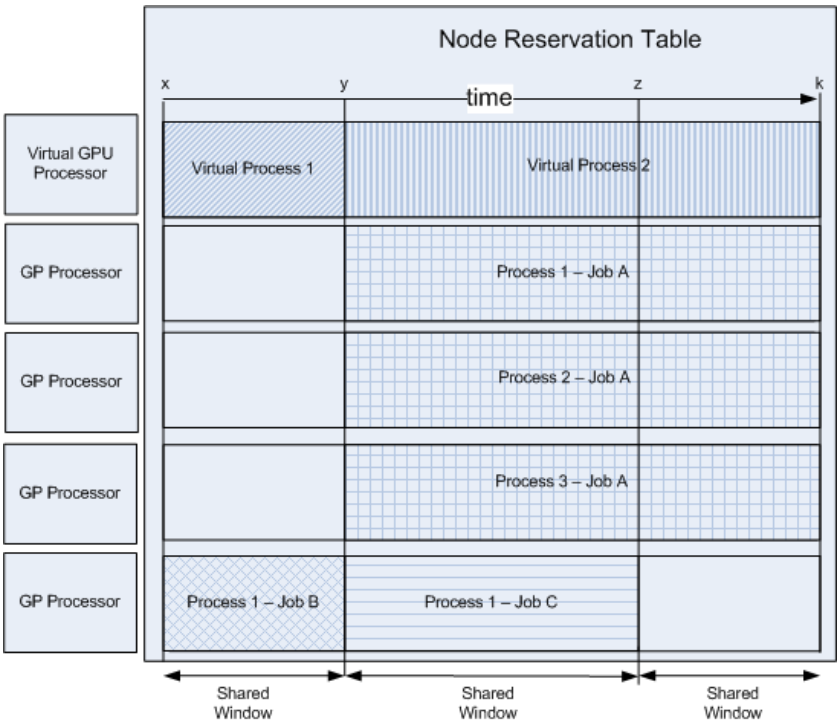


Figure 2: Reservation tables at node level

3) Modeling the computing power: The processor model that is used in the kento-perf simulator does not consider architecture details of the processing units. Thus, it does not implement a pipelined microprocessor neither models how the assembly instructions are executed. In this work, we have used two different processors properties: the frequency and the *CPU Factor*. The *CPU Factor* correlates the computing power of the different processors available in the system. Thus, the faster processors of the system will have a *CPU Factor* of 1. The other processors will have a value according to how they perform (on average) with respect to the faster one. In the system evaluated in this work, we have supposed that the GP computing elements have a *CPU Factor* of 1 and the GPU cores have a *CPU Factor* of 3. We are using these values because we consider that a GPU core will perform similar that an Out of Order processor (OoO). Thereby, as it is general assumed, the GP has an speedup of 3 versus and OoO processor.

Similar to the *CPU Factor*, we define a base frequency that is the fastest frequency used by all the processors in the system. It is called the baseline frequency B_{freq}. Each processor has associated a set of frequencies $(freq_1, \ldots, freq_n)$. The set of frequencies and voltage levels depends on the processor. However, for simplicity we have assumed that all the computing elements (including the GPU cores and GP processors) use the same frequency and voltage levels. The characterization of the frequency and voltage levels is presented in section III-D.

The formula presented below is used to compute the simulated job runtime for a given allocation. The original job runtime (Job_{ort}), is extracted from the workload used as a simulation input. The runtime α_{rt} for the job's process α, allocated in a processor cpu at frequency α_{freq}, is computed as follows:

$$Job_{rt} = \frac{B_{freq}}{\alpha_{freq}} \times Job_{ort} \times CPUFactor_{cpu}$$

B. The system architecture

We model a system based on a cluster architecture where each node is characterized by a set of computational elements, and a memory subsystem with a given capacity and bandwidth. In a given node all the processors share the memory bus. Two different types of computational elements compose a node. First, a node contains a set of GP processors. Second, some nodes can contain a set of GPU or many-core[3] computing elements. As we describe later, GPU processors provide high degree of parallelism but are limited by cores with less performance and by the memory bandwidth to the node provided by the socket. On the contrary, the GP processors provide access to processors with higher computational performance and memory bandwidth.

[3]For legibility we will use the *GPU* term for referencing both GPU and many-core processors.

C. The resource model

A computing node contains a set of n computing elements that share a given amount memory bandwidth (β). In the examples that are presented in this section, we use $n = 9$ and $\beta = 18$ GB/s as a matter of example. Other values can be used by the model in order to evaluate different configurations. According to the previous runtime model, in those intervals of time in which the total memory bandwidth demand is higher than β, a penalized runtime is added to each of the processes running to the GP processors of the node. The processes running in the GPU may experiment different penalties due to they are limited by the memory bandwidth provided by the socket to the node.

The node can potentially allocate m processes: $m - k$ processes running in the GP computing elements, and k processes running in the GPU computing element (where k is the number of cores of the GPU). However, we do not assume that the β memory bandwidth is equally shared among the m processes. The processes allocated to the GPU will only have access to the bandwidth provided by the socket for accessing the memory bandwidth. Thus, in our model, we assume that the GPU has socket to access the shared memory with a given bandwidth β_{gpu}, which is the maximum bandwidth of the GPU cores. In the architecture that we evaluate in this paper, we set β_{gpu} to $(\beta)/9$ GB/s. All the processes running in the GPU will be able to share only this amount of memory bandwidth, which is the bandwidth socket to the memory that the node provides. Thus, they may experiment a penalized time due to the contention of memory bandwidth modeled by the GPU reservation table presented previously.

The processes that are allocated to the GPU cores will experiment a slowdown in its execution of 3 due to the runtime penalty caused by the core performance. In our model, we assume that this slowdown will cause a reduction of 3 also in the memory bandwidth required by these processes. Obviously, not all the applications have this bidirectional implication. However, in order to simplify the evaluation of our approaches we consider that all the stream of jobs follow this behavior. Future work will consider workloads containing a different type of job topologies.

As an example, suppose an scenario with a node that has a total capacity of 18 GB/s memory bandwidth. A job (*job1*) has 8 processes running in the GPU and two processes running in two GP processors of the node. For simplicity, suppose that no other processes are running in the node, and the job is totally allocated in the node. Each process of *Job 1* would require a total amount of 4 GB/s of memory bandwidth in a GP processor. However, the processes running in the GPU will perform three times slower. Thereby, we consider that the amount of memory bandwidth required by each process will be reduced by a factor of three (each process will request 1.3 GB/s approximately). In this example, the total bandwidth that will be requested to the GPU socket will be 10 GB/s approximately. The GPU processes will be penalized by a factor of $10/4$. The other processes of *Job 1* running on the GP computing elements of the node will not experiment any penalty. This is caused due to the fact that the total memory bandwidth available in the node is 18 GB/s and the total required by the processes and the GPU is 12 GB/s.

D. Modeling the energy consumption

In order to estimate the energy consumed during the job execution, we have used the frequency voltage relation presented in [27]. Critical threads are those threads that have the larger runtimes with respect to the others due to memory contention. The non-critical threads run in lower frequency in order to finish at the same time that the critical threads. The model assumes that the critical thread finishes its execution in T units of time. A non-critical thread will finish its execution in $T - k$ units of time ($0 \leq k < T$). The energy consumed by the thread running at the maximum core frequency is modeled by the following formula:

$$E_{f_{max}} = c \times {V_{dd_{max}}}^2 \times f_{max} \times ((T - k) \times T)$$

Table I shows the different frequency and voltage levels which the processor can run at. In particular, this table corresponds to an Intel Processor XScale [34]. We have assumed that the processors and cores running in our system follow these voltage/frequency relations. V_{dd} (extracted from a voltage/frequency of table I) is the voltage that corresponds to the frequency used f_{max} during the thread's runtime. The *c* is the capacitance of the processor. Note that the model assumes that during the rest of time the thread will remain idle. If the frequency for the non-critical thread is scaled-down in order to synchronize its finalization with respect to T, the consumed energy is:

$$E_{f_{scaled}} = c \times n \times {V_{dd_{scaled}}}^2 \times f_{scaled} \times ((T - k) \times T)$$

The factor n is computed according to the new processes frequency. f_{scaled} and $V_{dd_{scaled}}$ are the new frequency and voltage at which the processes will run, respectively.

IV. The Scheduling Strategy

In this section, we present the scheduling strategy that is built on top of the model that has been introduced in the previous section. Its main goal is to schedule and allocate the jobs considering the heterogeneous characteristics of the system. But more important, it considers how the memory bandwidth is used by the different job processes. Thereby, it schedules based on how they can behave depending on: it requirements, the computing element where they can be allocated and the estimated bandwidth contention.

VOLTAGE FREQUENCY TABLE					
LEVEL	MV	GHZ	LEVEL	MV	GHZ
0	700	0.90	7	875	2.80
1	725	1.20	8	900	3.05
2	750	1.45	9	925	3.30
3	775	1.75	10	950	3.50
4	800	2.00	11	975	3.75
5	825	2.25	12	1000	4.0
6	850	2.35			

Table I: Voltage Frequency relation values used in the evaluation

A. The resource selection policies

We consider two different resource selection policies that target two different problems. On the one hand, the Less Consume policy tries to minimize the resource sharing contention considering the job requirements and the status of the system. On the other hand, the Power Aware resource selection policy tries to minimize the energy consumption of a given job allocation.

1) LessConsume policy: Using the model that we have presented in section III-A, we designed two new resource selection policies. First, the *LessConsume* that attempts to minimize the job runtime penalty that an allocated job will experience. Based on the utilization status of the shared resources in the current scheduling outcome and job resource requirements, the *LessConsume* policy allocates each job process to the free allocations in which the job is expected to experience the lowest penalties. Second, we designed the *LessConsume Threshold* resource selection policy which finds an allocation for the job that satisfies the condition that the estimated job runtime penalty factor is lower than the given value *Threshold*. It is a variant of the *LessConsume* policy and was designed to provide more sophisticated mechanisms to improve the efficiency of job allocations.

The core algorithm of the *LessConsume* policy is similar to the *First Fit* resource selection policy. This last one selects the first free N processors in time where the job can be allocated. However, in contrast to First Fit, in the *LessConsume* policy, once the base allocation is found, the algorithm computes the penalties associated to the different processes that would be allocated in the reservation table. Thereafter, it attempts to improve the allocation by replacing the selected buckets (used to create this initial allocation) that would have higher penalties with buckets that can be also selected, but that have not been evaluated. The *LessConsume* algorithm will iterate until the rest of the buckets have been evaluated or the penalty factor associated to the job is 1 (no penalty).

2) Power Aware resource allocation policy: The previous resource selection policy selects a set of processors and interval of time to allocate a job. Furthermore, it returns the estimated penalized time due to resource sharing saturation for each of the job processes. Using this information, the Power Aware resource selection policy selects the appropriate frequency for each processor according to the imbalance that is caused to the job processes due to the resource sharing contention and due to the hardware differences among the different computing elements. The main rationale of this policy is the following: if a process p_i of a job with a nominal runtime of k_{nrt} cannot finish after than $k_{nrt} * \beta$ due to the penalties (being $\beta > 1$ and being $k_{nrt} * \beta$ the maximum penalized process runtime), the runtime and memory bandwidth required for the rest of the processes will be reduced according to this β. Thereby, as the limiting factor between the original runtime and the new runtime for these no-critical processes is the memory bandwidth, the frequency used to run the processes can be updated to equilibrate the penalty caused by memory contention and computing power.

Given job α having n processes, the Power Aware (PA) policy is implemented with the following algorithm:

1) It selects the process $p_{critical}$ that has the higher estimated penalized runtime $T_{critical}$ (with runtime penalty $pRt_{critical}$). This time includes, the process runtime, the penalty due to resource contention and due to processor computing power (only associated to processes running on a GPU computing element).
2) For each process k in the allocation (including $p_{critical}$):
   a) If the penalized runtime p_k (including the memory bandwidth penalty $pt_{k,bw}$ and the computing power penalty $pt_{k,cp}$) is less than $p_{critial}$, the memory bandwidth required by the process is reduced by the factor
   $$rfact = \frac{pt_{k,bw} + pt_{k,cp}}{pRt_{critical}}.$$
3) At this point, all the required bandwidth for the process has been updated according to the $p_{critical}$ process. The penalized runtime due to memory contention is recomputed for all the other jobs, according to the new status of the reservation table.
4) For each process p_k in the allocation (including $p_{critixal}$):
   a) Computes the scale-down factor $\alpha_{SD,k}$ that the process should experiment in order to finish at the timestamp $t_0 + T_{critical}$
   $$SD_k = \frac{pt_{k,bw} + pt_{k,cp} + rt_k}{T_{critical}}$$
   b) Using the voltage/frequency table it selects the frequency $\alpha_{k,freq}$ that satisfies that

$$SD_k \approx \frac{freq_{max}}{freq_k} \; and \; \frac{freq_{max}}{\alpha_{k,freq}} \leq T_{critical}$$

5) Returns the list of frequencies associated to each process allocation.

The previous algorithm is divided in two phases: one to find the critical process and update the estimated memory bandwidth for all the processes according to its penalty, and the other to compute the scale-down factor frequency required by the new computation requirements of the rest of the processes. In the initial phase, the process that is critical for the job finalization is selected. Critical means that it is the process that will finalize it execution later for the given allocation. Following the same nomenclature that [27] its process runtime is called $p_{critical}$. The memory bandwidth required by the rest of processes that have lower penalized runtimes is reduced according to the factor $rfact$. We assume that if a given process cannot achieve a speedup of 1 with respect to its nominal runtime (original workload runtime), caused due to computational capacity and memory bandwidth contention for the critical thread, the rest of the processes will also experiment the same speedup reduction. Note that we may consider that each process is independent; thereby the memory required would not be updated. This will be part of future work studies. In the second phase, the runtime for the rest of the processes is extended in order to match the finalization of the critical process. This is done by using the appropriate frequency that satisfies that applied to the original runtime it is scaled to the $T_{critical}$. Note the new scaled runtime cannot be higher than $T_{critical}$, thereby if two frequencies can be used to satisfy the scale-down factor, the higher frequency will be used.

As example, suppose an scenario with a job (*Job 1*) that has six different processes (*process 1*,..., *process 6*). The first three processes will be allocated to the GPU and will have the highest estimated penalty time due to resource sharing contention and computing power penalty. In the initial phase, the first process of processes *1, 2* or *3* is selected as $p_{critical}$ (all of them have the same estimated penalty). Next, the estimated memory bandwidth required by the rest of the processes is update according to the $p_{critical}$. For instance, *process 6* that is running in the node *R*, has a penalized runtime of *po6* and it is reduced to *pu6* due to the new estimated memory bandwidth achievable due to the new runtime. Processes *4* and *5* are allocated to the GP computing elements of the same node and will have less estimated runtime penalty. Finally, *process 6* will be allocated to *Node k* having the lowest estimated penalty of *Job 1*'s processes. The Power Aware resource selection policy will estimate at which frequency has to run each process, according to the computing element where is allocated, and the associated penalty. The goal is to make all the processes finish at the same time. In this example, processes *4* and *5* will run at 1.75 GHz and *process 6* will run at 0.75 GHz. This mechanism allows the process that is not allocated to the GPU to save energy (see section III-D).

V. Evaluation Methodology

In our experiments we have used the cleaned versions of thee different workloads from the Parallel Workload Archive [35]:

- The San Diego Supercomputer Center (system with 416 processors) workload, henceforth referenced as SDSC [36]. It is characterized by containing jobs with relatively small degree of parallelism (almost al of them use less than 16 processors) and with very small runtimes (80 % of the jobs have a runtime smaller than 10 minutes).
- The Cornell Theory Center (system with 512 processors) workload, henceforth referenced as CTC [37]. It is characterized by containing serial jobs with relatively larger runtimes (compared to the previous workload). Fifty percent of the jobs have runtimes lagers than 10 minutes. Furthermore, more than 20% of them have a runtime higher than one hour.
- The SDSC Blue Horizon (system with 1152 processors) workload, henceforth referenced as SDSC-BLUE. It is characterized by containing mainly a stream of jobs that require 8 (50% of the jobs), 16 (20% of the jobs), 64 (15% of the jobs) and 256 processors. Furthermore it does not contain sequential jobs. Concerning the job runtime, approximately 60% of the jobs have a runtime smaller than 10 minutes, 20% of them higher than 1 hour and some fraction of them around four days.

For each of this workloads we have defined a system composed of two types of nodes: nodes of 6 GP processors and one GPU of 32 cores, and nodes with 10 GP processors. Both type of nodes have 64 GB/s of memory bandwidth. We set up these values due according to the system that we are envisioning. In the experiments we evaluate the impact of using the techniques presented in this paper within different backfilling strategies.

In order to evaluate the impact of different memory pressures on the performance of our strategy, we have defined four different types of memory workloads. Each of these workloads contains a different percentage of job memory bandwidth demand. The two first columns of table II present the different job types that we have used in our experiments, and the memory (in GB/s per process) that each job type requires. The other four columns show the four workload types that we have evaluated. Each of them has different percentages of each job type depending on the workload type. The first one is memory bounded (almost all the workload jobs require high amount of memory bandwidth per process). The second one is computational capacity bounded (almost all the jobs require a relatively small amount of memory bandwidth per process). The third

WORKLOAD CHARACTERIZATRION					
JOB TYPE	MEM. DEMAND	MEM.	HYBRID	COMP.	EQUIP.
0	64 GB/s	10	5	5	16.6
1	32 GB/s	20	15	5	16.6
2	16 GB/s	40	15	20	16.6
3	8 GB/s	20	35	20	16.6
4	2 GB/s	5	15	30	16.6
5	1 GB/s	5	15	20	16.6

Table II: Percentage of job topology per workload type

one is the hybrid version of the two previous ones. Finally, the last workload has an equi-probability to contain jobs of any of the presented job topologies.

The resource contention model is considered in all the experiments that we have conducted. In particular, we have evaluated the EASY-Backfilling scheduling policy using the following resource selection policies:

- LessConsume plus Power Aware **(LC-PA)**
- First-Fit plus Power Aware **(EASY-PA)**
- First-Fit **(EASY-REG)**

The *LessCosume* resource selection policy uses the estimated memory bandwidth demand and the requested runtime to carry out the scheduling. In order to evaluate the impact of the estimation accuracy on different metrics, we have evaluated three different bandwidth estimations (10%, 50% and 100% of estimated error with respect to the real required memory bandwidth) for each scenario. In future work, we will include a real prediction models in the simulator in order to evaluate their impact on the performance of the system (i.e: [38][39][40]).

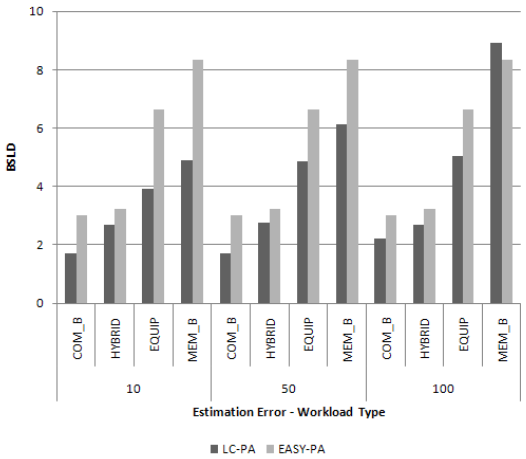
VI. RESULTS

Figure 3 presents the average and the 95_{th} percentile of the bounded slowdown (BSLD) obtained in the evaluation from the different scenarios. Each plot shows the obtained BSLD using the LC-PA and EASY-PA policies within the different memory workloads and memory bandwidth estimation errors. The BSLD is a good metric to show the performance of the system. Note that the performance for the EASY-PA policy is the same for all the different BW estimations because it does not use this information to carry out the scheduling.

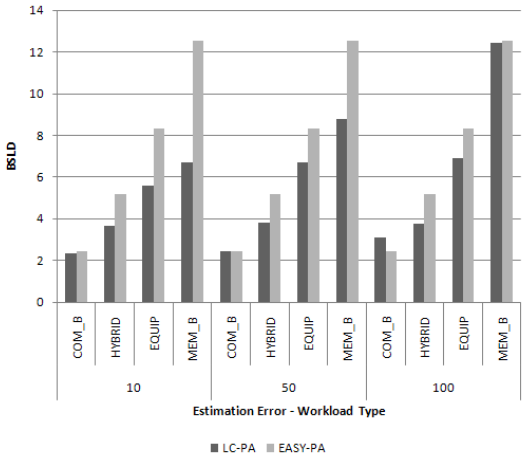
The analysis of the LC-PA policy shows that GPU cores improve the system performance when the workload is composed by fewer jobs with relatively low memory bandwidth requirements. LC-PA policy is able to perform efficient allocations if it considers the estimated memory bandwidth demand. In these situations, the scheduling strategy tries to allocate these jobs to the GPU cores, and the other GP processors are used by those applications that have higher memory bandwidth requirements. This fact provides clear advantages to these policies with respect to the EASY-PA policy, which allocates the jobs based only on their runtime. Specifically, allocating jobs based only on their runtime may result in inefficient scheduling decisions, such as allocating applications with high memory bandwidth demand to the GPU. Moreover, in this scenario the LC-PA policy has higher probability to backfill those jobs with less memory bandwidth demand to the GPU.

When the workload is composed by a higher number of jobs with high memory demand, GPU cores cannot enhance the system performance. In this scenario, jobs that are allocated to the GPU suffer high runtime penalties and there is an important degradation in their performance with both policies. The EASY-PA scheduling policy may carry out inefficient scheduling decisions (inefficient allocations of processes to resources) due to the unconsciousness of how the memory bandwidth is used. In some scenarios, LC-PA and EASY-PA provide similar performance levels. For instance, for the CTC system with the scenario with the memory bounded workload and 100% of estimation error in the memory bandwidth, the average BSLD is lower using the EASY-PA strategy.

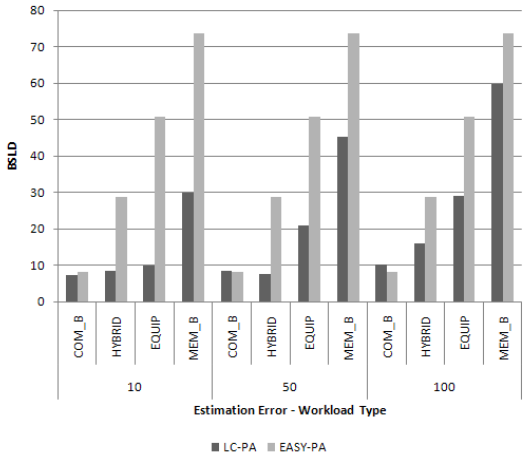
We have observed that the scheduling strategies proposed in this work provide better performance results when the memory bandwidth estimations are accurate. In all the scenarios, increasing the error results in a considerable increase of the 95th percentile of the BSLD obtained with LC-PA with respect to the results obtained with the EASY-PA policy. We have stated that in these scenarios the PA assigns inefficient scale-down factors. In many situations, assigning lower frequencies to processors resulted in performance degradation due to memory bandwidth saturation. This fact becomes critical in those scenarios where the number of memory bounded jobs is high. For instance, in the SDSC-Blue workload, the memory bounded workload with 100% of estimation error, the LC-PA policy has an BSLD of 193, while the EASY-REG has an average BSLD of 180.



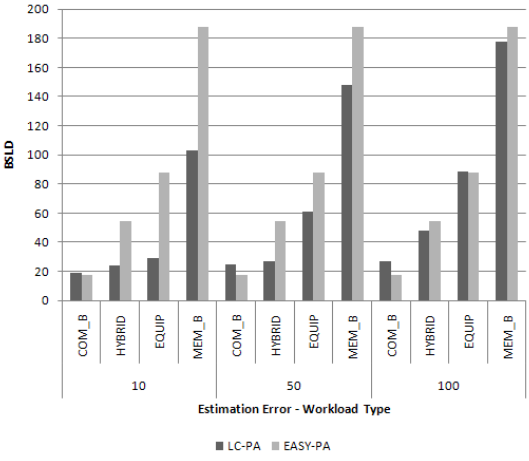
(a) BSLD (Average) - CTC



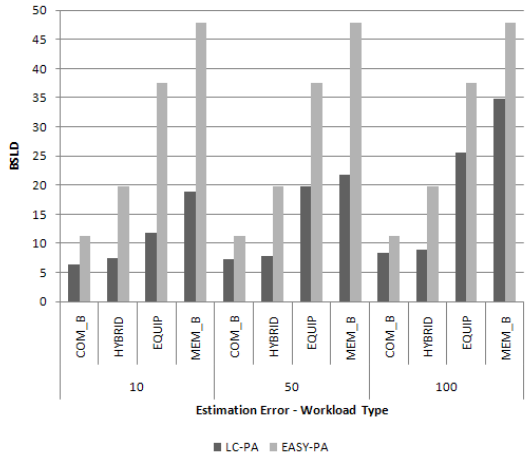
(b) BSLD (Percentile 95th) - CTC



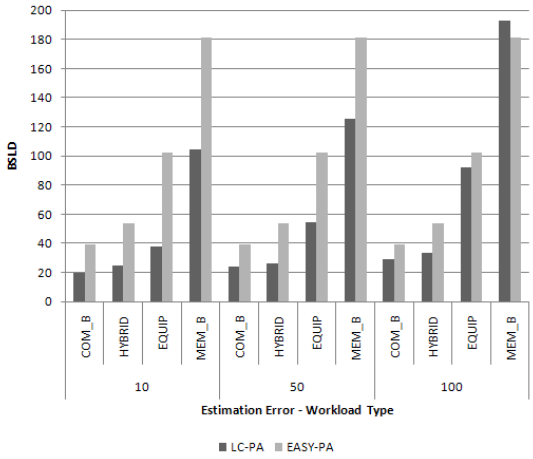
(c) BSLD (Average) - SDSC



(d) BSLD (Percentile 95th) - SDSC



(e) BSLD (Average) - SDSC-BLUE



(f) BSLD (Percentile 95th) - SDSC-BLUE

Figure 3: Performance Results

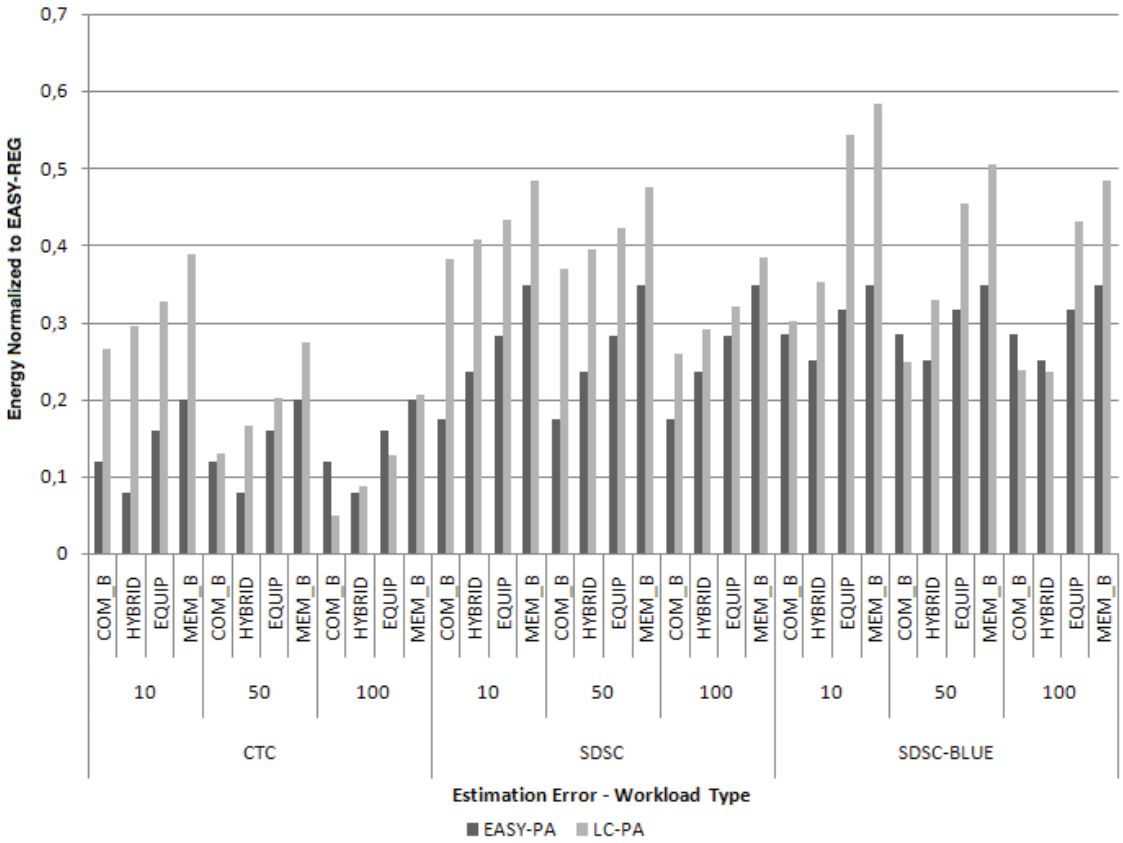


Figure 4: Relative energy savings

However, in the same scenario but with an error of 10%, BSLDs are 104 and 181 respectively.

These results state that having an accurate estimation model for the job memory requirements is crucial to deploy our scheduling strategy successfully in production systems. If job memory requirements are not sufficient accurate, jobs with relatively high memory bandwidth demand may be allocated to the GPU; therefore, system performance may be penalized dramatically.

Figure 4 shows the percentage of energy saved with our policies for each scenario with respect to the energy consumed with the EASY-REG policy. In all the different scenarios the LC-PA scheduling strategy saves more energy than the EASY-PA. In general, in those scenarios where the workload is composed by jobs

with high memory bandwidth demand and computational bounded jobs, the LC-PA policy is able to achieve better energy savings. A clear example is presented in the SDSC-BLUE workload with a 10% of error estimation. Here the LC-PA policy is able to save up to 33% of energy consumption with respect to the EASY-REG in the COM_B and HYBRID workloads. However, when the stream of jobs is composed by higher number of memory bounded jobs, this improvement increases to 50% and 58% in the EQUIP and MEM_B workloads, respectively. The LC-PA policy is able to better allocate jobs than the EASY-REG and the EASY-PA. However, as we stated in the system performance, when the memory bandwidth demand and the estimation error are higher (memory bounded and equi-probable workloads), the results obtained with LC-PA and EASY-PA are closer. This fact is clearly stated in the the SDSC workload. Here the difference between the two policies is less than 10% in all the workloads using a 100% of estimation error.

We also have stated that energy savings are substantially reduced when the stream of jobs is characterized by large jobs. In this case, the increase of bandwidth estimations errors makes the errors to become more critical. Thereby, when inefficient scheduling decision are taken, the subsequent scheduling decisions may be dramatically affected. This is directly translated to higher slowdowns and less energy savings. This fact can be visually observed in the plot of energy savings (figure 4) for CTC. For example, the energy saving with respect to the EASY-REG is reduced from 39% in the MEM_B workload and an estimation error of 10% to 20% in the same workload type but with an error of 100%. Thereby, in systems where jobs are relatively large, the accuracy of the memory bandwidth estimation is crucial

when applying the proposed techniques.

Results indicate that traditional backfilling policies can improve the energy savings over 30% using the PA resource selection policy. For example, in the SDSC-BLUE workload, using a 110% estimation error and the MEM_B workload, the traditional EASY Backfilling strategy is able to save up to 37% of energy using the Power Aware resource selection policy. This is specially interesting because it implies that energy consumption can be optimized in HPC centers by adding the proposed resource selection policies to the scheduling strategy rather than changing the whole scheduling systems.

VII. Conclusions and Future Work

In this paper, we have described and evaluated a scheduling strategy for GPU and many-core architectures for HPC environments. Specifically, our strategy is a variant of the backfilling scheduling policy that considers the differences in terms of computational capacity and memory bandwidth of the GP processors and the GPU. Our strategy also considers the usage of DVS to reduce the energy consumption when processes do not need all the GP computing power. Our work also considers the impact of memory bandwidth contention of the socket from the GPU to the node and lower computational capacity of the GPU cores. Thus, our model also considers the usage of DVS when different processes are unbalanced due to memory bandwidth contention and computational capacity imbalance. Bandwidth contention is considered at the level of the GPU computing element socket and at the node memory bandwidth. The performance studies reveal that the backfilling variants proposed in this paper can provide improvements in the performance of up to 30% with respect to the traditional backfilling strategies. However, the results show that the memory estimation provided by the user has to be accurate when the stream of jobs submitted to the system is memory bounded. Our evaluation shows that the scheduling strategies proposed in this work are able to save over 40% of energy consumption with respect to EASY-Backfilling. The results also show that there are different aspects of EASY Backfilling that can be improved in order to reduce the energy consumption of HPC systems.

As a part of our future work, we will investigate the impact of using different types of job topologies on performance and energy consumption. The main goal is to add to the simulation and runtime model the capability to model jobs where the processes have dependencies; for instance, modeling OpenMP and MPI jobs. The second short term goal is to add to the simulator a memory bandwidth prediction model and use it rather than the estimation that is provided by the user. Since many research works have been done in this area (i.e: [38][39][40]), the main idea is to evaluate the impact of the proposed prediction techniques on our scheduling strategy.

Acknowledgements

This paper has been supported by the Spanish Ministry of Science and Education under contract TIN200760625C0201.

References

[1] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "Gpu cluster for high performance computing," ACM/IEEE Conference on Supercomputing, p. 47, 2004.

[2] Z. Fan, F. Qiu, and A. Kaufman, "Zippy: A framework for computation and visualization on a gpu cluster," Computer Graphics Forum, vol. 27, no. 2, pp. 341–350, 2008.

[3] M. Showerman, J. Enos, A. Pant, and V. Kindratenko, "Qp: A heterogeneous multi-accelerator cluster," International Conference on High-performance Clustered Computing, 2009.

[4] V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, and W. Hwu, "Gpu clusters for high-performance computing," Workshop on Parallel Programming on Accelerator Clusters, pp. 1–8, 2009.

[5] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," ACM Trans. Graph., vol. 27, no. 3, pp. 1–15, 2008.

[6] "Nvidia's next generation cuda compute architecture: Fermi," 2009, NVIDIA Corporation.

[7] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," IEEE Micro, vol. 28, no. 2, pp. 39–55, 2008.

[8] T. Hamano, T. Endo, and S. Matsuoka, "Power-aware dynamic task scheduling for heterogeneous accelerated clusters," IEEE International Symposium on Parallel and Distributed Processing, pp. 1–9, 2009.

[9] F. Guim, J. Corbalan, and J. Labarta, "Modeling the impact of resource sharing in backfilling policies using the alvio simulator," IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 145–150, 2007.

[10] D. G. Feitelson and B. Nitzberg, "Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860," Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 949, pp. 337–360, 1995.

[11] D. G. Feitelson, "Packing schemes for gang scheduling'," Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 1162, pp. 89–110, 1996.

[12] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 1459, pp. 1–24, 1998.

[13] D. G. Feitelson, "Workload modeling for performance evaluation," Performance Evaluation of Complex Systems: Techniques and Tools, Lect. Notes Comput. Sci., vol. 2459, pp. 114–141, 2002.

[14] D. Tsafrir and D. G. Feitelson, “Instability in parallel job scheduling simulation: the role of workload flurries,” IEEE International Symposium on Parallel and Distributed Processing, 2006.

[15] M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera, “A hierarchical approach to workload characterization for parallel systems,” Performance Computing and Networking, Lect. Notes Comput. Sci., vol. 919, pp. 102–109, 1995.

[16] M. Calzarossa, L. Massari, and D. Tessera, “Workload characterization issues and methodologies,” Performance Evaluation: Origins and Directions, Lect. Notes Comput. Sci., vol. 1769, pp. 459–482, 2000.

[17] W. Cirne and F. Berman, “A comprehensive model of the supercomputer workload,” IEEE International Workshop Workload Characterization, pp. 140–148, 2001.

[18] ——, “A model for moldable supercomputer jobs,” IEEE International Symposium on Parallel and Distributed Processing, p. 10059.2, 2001.

[19] K. C. Sevcik, “Application scheduling and processor allocation in multiprogrammed parallel processing systems,” Performance Evaluation, vol. 19, no. 2-3, pp. 107–140, 1994.

[20] A. B. Downey, “A parallel workload model and its implications for processor allocation,” IEEE International Symposium on High Performance Distributed Computing, p. 112, 1997.

[21] J. Skovira, W. Chan, H. Zhou, and D. A. Lifka, “The easy - loadleveler api project,” Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 1162, pp. 41–47, 1996.

[22] D. Tsafrir, Y. Etsion, and D. G. Feitelson, “Backfilling using system-generated predictions rather than user runtime estimates,” IEEE Trans. Parallel Distrib. Syst., vol. 18, no. 6, pp. 789–803, 2007.

[23] ——, “Backfilling using runtime predictions rather than user estimates,” Technical Report 2005-5, School of Computer Science and Engineering, The Hebrew University of Jerusalem, 2005.

[24] S. H. Chiang, A. C. Arpaci-Dusseau, and M. K. Vernon, “The impact of more accurate requested runtimes on production job scheduling performance,” Workshop on Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 2537, pp. 103–127, 2002.

[25] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, “Parallel job scheduling - a status report,” Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 3277, p. 9, 2004.

[26] G. Dhiman, K. K. Pusukuri, and T. Rosing, “Analysis of dynamic voltagescaling for system level energy management,” USENIX HotPower’08, 2008.

[27] Q. Cai, J. Gonzlez, R. Rakvic, and G. Magklis, “Meeting points: using thread criticality to adapt multicore hardware to parallel regions,” International Conference on Parallel Architectures and Compilation Techniques, pp. 240–249, 2008.

[28] A. Merkel and F. Bellosa, “Memory-aware scheduling for energy efficiencyon multicore processors,” USENIX HotPower’08, 2008.

[29] B. Lawson and E. Smirni, “Power-aware resource allocation in high-end systems via online simulation,” International Conference on Supercomputing, pp. 229–238, 2005.

[30] K. H. Kim, R. Buyya, and J. Kim, “Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters,” IEEE International Symposium on Cluster Computing and the Grid, pp. 541–548, 2007.

[31] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, “Energy-aware high performance computing with graphic processing units,” Workshop on Power Aware Computing and Systems, 2008.

[32] F. Guim, J. Corbalan, and J. Labarta, “Resource sharing usage aware resource selection policies for backfilling strategies,” High Performance Computing & Simulation Conference, 2008.

[33] F. Guim, I. Rodero, and J. Corbalan, “The resource usage aware backfilling,” Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci., vol. 5798, pp. 59–79, 2009.

[34] A. Varma, E. Debes, I. Kozintsev, and B. Jacob, “Instruction-level power dissipation in the intel xscale embedded microprocessor,” Annual Symposium on Electronic Imaging Science and Technology, 2005.

[35] D. G. Feitelson, “Parallel workload archive,” 2009. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[36] K. Windisch, V. Lo, D. Feitelson, R. Moore, and B. Nitzberg, “A comparison of workload traces from two production parallel machines,” Symposium on the Frontiers of Massively Parallel Computation, p. 319, 1996.

[37] D. G. Feitelson and L. Rudolph, “Workload evolution on the cornell theory center ibm sp2,” Job Scheduling Strategies for Parallel Processing, Lect. Notes Conput. Sci., vol. 1162, pp. 27–40, 1996.

[38] E. Ipek, B. D. Supinski, M. Schulz, and S. McKee, “An approach to performance prediction for parallel applications,” Euro-Par, pp. 196–205, 2005.

[39] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, “Methods of inference and learning for performance modeling of parallel applications,” ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 249–258, 2007.

[40] Z. Wang and M. F. O’Boyle, “Mapping parallelism to multicores: a machine learning based approach,” SIGPLAN Not., vol. 44, no. 4, pp. 75–84, 2009.