The Journal of Supercomputing (2022) 78:200–221
https://doi.org/10.1007/s11227-021-03883-5

Scheduling for heterogeneous systems in accelerator-rich environments

Serif Yesil[1] · Ozcan Ozturk[2]

Accepted: 11 May 2021 / Published online: 25 May 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

The world is creating ever more data and the applications are required to deal with ever-increasing datasets. To process such datasets heterogeneous and manycore accelerators are being deployed in various computing systems to improve energy efficiency. In this work, we present a runtime management system designed for such heterogeneous systems with manycore accelerators. More specifically, we design a resource-based runtime management system that considers application characteristics and respective execution properties on the nodes and accelerators. We propose scheduling heuristics and run time environment solutions to achieve better throughput and reduced energy in computing systems with different accelerators. We give implementation details about our framework; show different scheduling algorithms, and present experimental evaluation of our system. We also compare our approaches with an optimal scheme where integer linear programming approach has been implemented for mapping applications on the heterogeneous system. While it is possible to extend the proposed framework to a wide variety of accelerators, our initial focus is on Graphics Processing Units (GPUs). Our experimental evaluations show that including accelerator support in the management framework improves energy consumption and execution time significantly. We believe that this approach has the potential to provide an effective solution for next generation accelerator-based computing systems.

Keywords Scheduling · Energy · Heterogeneous computing · Accelerator · Graphics processing unit · GPU

✉ Ozcan Ozturk
ozturk@cs.bilkent.edu.tr

Serif Yesil
s.serifyesil@gmail.com

[1] Computer Science Department, University of Illinois at Urbana-Champaign, Champaign, IL, USA

[2] Department of Computer Engineering, Bilkent University, Çankaya/Ankara, Turkey

1 Introduction

Computing systems are required to deal with ever-increasing datasets as the number of applications and users increases. While mankind created data was 150 exabytes (billion gigabytes) in 2005 and 1200 exabytes in 2010, it is expected to be beyond 35,000 exabytes in 2020. For example, astronomical image data from ALMA (the Atacama Large Millimeter/submillimeter Array) [1] is 1 terabyte per day, or Google processed about 24 petabytes of data per day in 2008 [9]. Big Data term is used for such systems, where complexity is in three dimensions, namely volume, velocity, and variety [20]. Different kinds of data from different sources appear mostly in an unstructured form (95% of created information was unstructured in 2010), and the data sizes are moving from terabytes to zettabytes. Moreover, streaming data makes it impossible to store all the data produced. Therefore, it is necessary to design systems, where software libraries with distributed processing capabilities are available and analysis of big data problems across different kinds of hardware resources is possible.

To support such systems, it is inevitable to have the hardware infrastructure. According to the National Institute of Standards and Technology (NIST), current computing systems in the cloud are defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [41]. Storage devices, networks, servers, applications, and services are components of computing systems, where current high-end computing systems consist of generations of servers with different processing, network, and storage capabilities [4]. At the same time, these systems are getting more and more complex with improved accelerator technologies. Since manycore accelerators are being deployed in the high-end computing systems [16], heterogeneity in these systems rapidly increases. Dealing with heterogeneity in such systems is not a trivial task because the infrastructure is shared among multiple applications and multiple users [22] in the system.

Traditionally, data-intensive applications are executed in single instruction multiple data (SIMD) environments in a massively parallel fashion. However, with the advances in chip manufacturing technology, manycore accelerators [18, 21] have been developed and used in desktops, servers, and clusters since the first debut of NVidia's GeForce [18]. They even made their way into embedded systems with the widely used NVIDIA Tegra [42]. Manycore accelerators are also being deployed in the Cloud computing systems [17] and are expected to be in the core of Big Data computing systems in the coming years. While GPGPUs (General Purpose GPUs) such as NVidia's Fermi [29], AMD's AMD Radeon HD 6990 [3] are widely used in the current clusters, other technologies such as Intel Many Integrated Core (MIC) Architecture [21] is also being used.

While the number of accelerator technologies used in computing systems increases, Big Data applications that process millions of terabytes of data does not exploit the available accelerators. This is mainly due to the fact that using these accelerators for Big Data applications is not a trivial task as accounting for heterogeneity in the system becomes more difficult. This is especially true for

utilizing the manycore accelerators as they exhibit different characteristics on different applications.

The main objective of the proposed work is to develop a runtime management system for parallel applications that can also be used in the context of Big Data applications running on computing systems with different accelerators. More specifically, we design a resource-based runtime management system that considers application characteristics and respective execution properties on the nodes and accelerators. We propose scheduling heuristics and run time environment solutions to achieve better throughput and reduced energy consumption in computing systems with different accelerators. More specifically, we suggest a heterogeneous computing runtime management system called Resource Based Scheduling System which aims to:

- Implement resource-based job scheduling heuristics in order to satisfy user resource requests for a job, distinguish CPU and accelerator jobs on the system, and decide which node to execute the job for execution time and energy efficiency.
- Create an extensible framework in which adding and removing nodes or accelerators are easy.
- Develop a scheduling heuristic that enables the execution of accelerator jobs on CPU and vice-versa when both accelerator and CPU executions are possible.

The rest of the paper is organized as follows. In the next section, we give a detailed comparison of our approach with the prior related efforts. In Sect. 3, we give the details of our resource-based scheduling system. ILP formulation is given in Sect. 4. Sect. 5 presents the experimental setup and the results. The paper is concluded in Sect. 6 with a summary of our major observations.

2 Related work

Many researchers addressed the problems of heterogeneous systems in the past. One research direction that is widely studied is to integrate already existing scalable massively parallel computing frameworks to heterogeneous systems. As expected, MapReduce is the first framework that comes to mind which is being widely used for heterogeneous systems to enable hybrid computing [39, 49].

In [36, 37], the authors proposed and evaluated heterogeneity in separate clusters with different types of processing nodes. While the approaches proposed in [36, 37] provide some level of heterogeneity, our work focuses on heterogeneity in two dimensions: (1) heterogeneity in single node and (2) heterogeneity among multiple nodes of the system. Previous work also addresses heterogeneous system scheduling. The most significant examples of heterogeneity are presented in [34, 35].

In a more recent study, authors in [13] propose a task scheduling strategy based on a genetic algorithm for CPU-GPU heterogeneous computing platforms. Bao et al. [5] propose a dynamic task scheduling stratgy for Heterogeneous System Architectures (HSA) and evaluate their approach on data-parallel applications. Authors in

a more recent study propose a feedback-based dynamic and elastic task scheduling scheme [45]. They aim to provide a better load balance and device utilization but with lower scheduling overhead. Liu et al. [23] propose a hardware-software design to minimize the energy cost of datacenters by a deadline-aware energy-efficient task scheduling algorithm. In [12], authors propose a task scheduling technique for heterogeneous computing platforms. Their main objective is to focus on heterogeneity with multicore CPUs and different accelerators like FPGAs.

An open-source software Torque [43] is also used in similar environments, which supports CPU-accelerator heterogeneous systems. However, Torque's mechanisms use GPU in a dedicated way which only provides GPU computing on-demand. As also stated in [35], Torque cannot exploit the capabilities of a framework like OpenCL. Besides heterogeneous systems, scheduling on homogeneous systems is also discussed widely [14, 36, 37].

In standard Open MPI [31] there is no direct support for heterogeneous nodes. If the nodes are not oversubscribed, then the scheduling takes place in a round robin fashion. Otherwise, depending on the *mpirun* call, MPI uses *slots*, i.e., the number of processors on each host. Schedulers (such as Slurm, PBS/Torque, SGE, etc.) use two scheduling policies, one by slot and one by the host. The default scheduling policy is by slot, where all slots are exhausted before proceeding to the next node. The other alternative is to schedule a single job on each node in a round-robin fashion. Communication layers such as the Byte Transport Layer (BTL) and the BTL Management Layer (BML) in MPI are not aware of processor heterogeneity [19]. In an effort to provide some level of heterogeneity support, authors in [50] propose a task scheduling approach based on MPI and CUDA by taking into account the node-level heterogeneous characteristics.

Kapil et al. [10] propose techniques to distribute OpenCL kernel workloads during run-time onto different devices according to power limitations and CPU load conditions. Their main focus is to achieve power-aware scheduling on CPU-GPU environments. On the other hand, Shulga et al. [40] propose a scheduler that selects targets to execute according to machine learning-based training with different data sizes. In [30], authors propose visual analysis techniques to evaluate the execution time of high-performance applications on hybrid architectures. GPUShare [15] is a middleware solution for achieving fair sharing among different GPU processes. Chen and Lee [8] propose G-Storm, a scheduling algorithm that targets Storm big data platforms. Specifically, they implement a scheduler for Storm which takes the GPU capacity into consideration. Bellavista et al. [7] propose a priority-based resource scheduling for distributed stream processing systems.

Other related efforts [6, 11, 24, 27, 44, 48] implement similar techniques on different platforms or using different parameters. As opposed to these previous contributions, we propose a new scheduling heuristic in our proposal. Firstly, our scheme considers submitted jobs to port between accelerators and CPU only after it decides that the impact on the application execution time will be the minimum among the available jobs in the system. And secondly, our heuristic considers resource decays in terms of user-requested resources when a node is idle for some iterations of the scheduler and uses this approach to maximize resource utilization while trying to increase throughput.

3 Resource based scheduling system

Our runtime system is designed to run on various distributed computing system settings. In such a framework, it is assumed that any node in the system is not aware of the rest of the system; rather, the system-level scheduler named as the Cloud Application Manager (CAM), controls the device allocation and utilization. As shown in Fig. 1, a hierarchical management system is envisioned. While in our current system we only have one level, this can easily be extended to a deeper management scheme with many nodes.

The access mechanisms to use the underlying CAM management system is abstracted by an access layer as shown in Fig. 2. Any higher level access request will be provided through available system calls. While we have implemented the necessary system calls to access the CAM module, some changes may be required to work for a specific cloud interface.

CAM communicates with the nodes in the system via Portable Operating System Interface (POSIX) sockets and stores all necessary information about the system's state. The state information is kept for: 1)Currently running jobs, 2)Jobs waiting to be scheduled, and 3)Node information. For the *running jobs*, we keep where the job is currently executing on, what kind of resource (CPU, GPU, etc.) it is using, the memory requirements, how long has it used the resource, expected duration, etc. Similar to running jobs, for the *jobs to be scheduled*, we keep the arrival time of the job, the expected execution time, what kind of resource it requires, etc. Execution times are either declared by the user or are estimated through historical data or profiling. For each node, the system keeps track of the number of cores (for CPU)

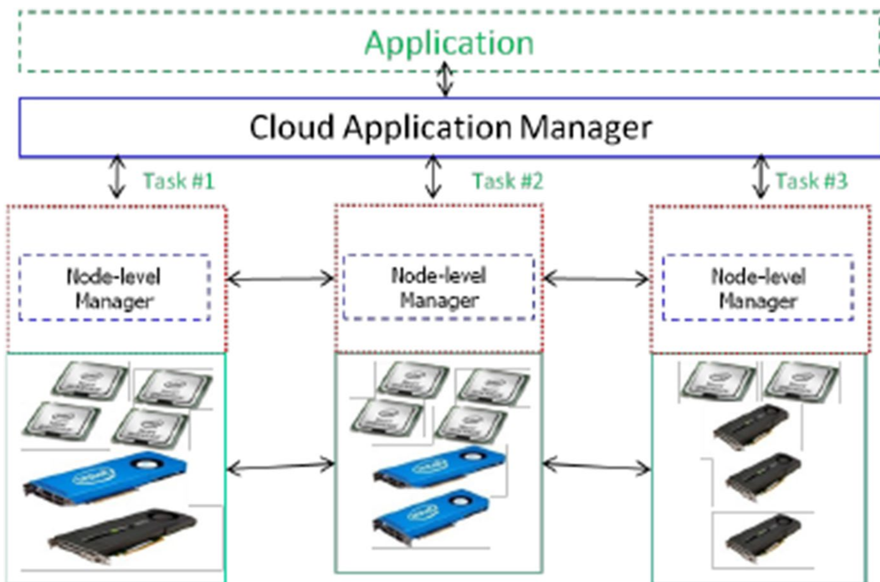


Fig. 1 Overview of a heterogeneous cluster management system

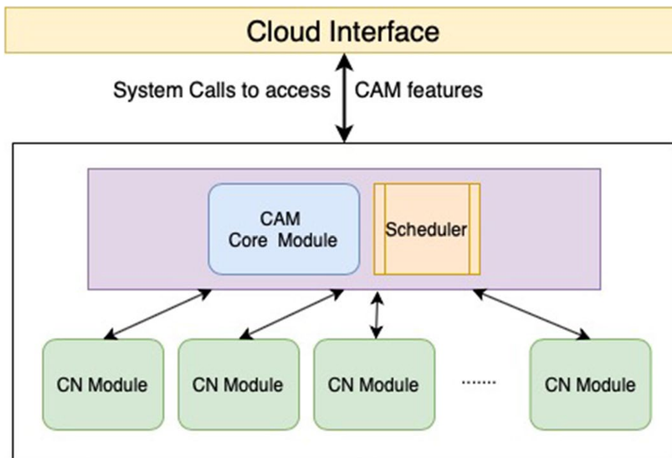


Fig. 2 The access mechanisms to use the underlying CAM management system is abstracted by an access layer

and the number of SMs (for GPU) as resource types. The voltage/frequency levels of cores, and CPU/GPU utilization for each available resource in the node is also kept. We assume that each node has one multiprocessor unit and multiple heterogeneous accelerators. In this environment, we focus on OpenMP [33] and OpenCL [32] applications. While there are previous efforts [26, 38] to use these high-level languages in cloud settings, they are also widely used in heterogeneous clusters.

It is usually common to assume such an environment since running parallel computing applications on a geographically distributed cloud system using multiple nodes will suffer from data transfer delays [51]. In addition to data transfer delays, heterogeneity will also create a bottleneck for a computing system. Our main concerns are energy consumption, utilization of the system, and executing applications on accelerators in the system. To achieve this, we developed resource-based job scheduling heuristics to satisfy user-level resource requests for jobs, while considering the different CPU/accelerator jobs on the system. Our heuristics mainly decide which node to execute the job and on which processing element.

We also implement scheduling heuristics that explore executing some of the accelerator jobs on CPU when both accelerator and CPU versions of applications are available for better energy consumption and execution time results. In order to implement these scheduling and mapping policies, we extended our system to handle OpenCL applications that can be converted into CPU applications easily.

Our framework is composed of two main components: cloud application manager (CAM) and Compute Nodes (CN). Our CAM implementation resides in a server where it controls different CNs while keeping the current state information about the system. According to the resources and requests, CAM decides which job to be executed on which node. An overview of the system can be seen in Fig. 3. CAM checks all nodes to determine which of the CNs are available for executing a new job. More specifically, CAM keeps a list of all jobs to keep the state of the jobs as

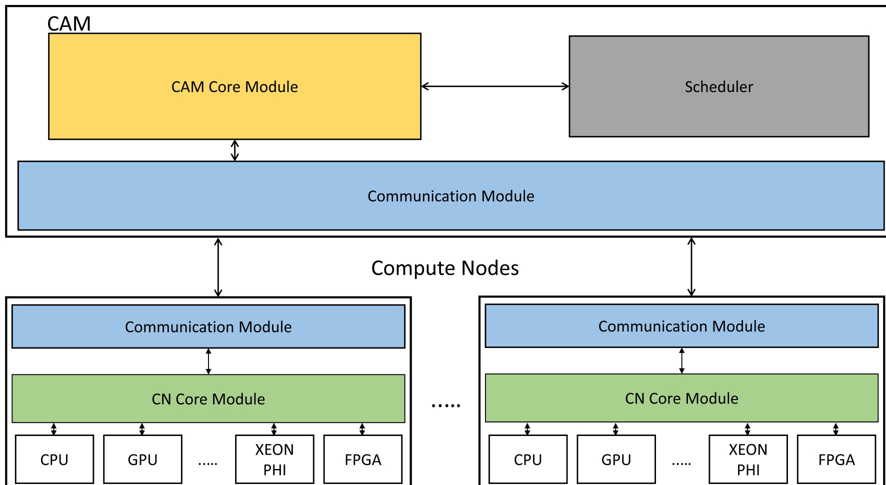


Fig. 3 System overview. Cloud application manager (CAM) and compute nodes (CN) are the main components

either running or waiting. CAM informs CNs about which particular jobs that will be executed next. After a CN finishes the execution of an assigned job, the CN notifies the CAM that CN is available for the next job. Meanwhile, CAM listens for user commands such as job submission, addition/deletion of a CN, and setting CAM's properties.

3.1 System overview

3.2 Scheduling

In order to enable an effective scheduling policy for CAM, we have implemented five different scheduling heuristics. These are RB-FIFO (resource-based first-in-first-out), RB-LJF (resource-based longest job first), RB-SJF (resource-based shortest job first), RB-LJF-BA (resource-based longest job first with bottleneck avoidance), and RB-LJF-OCL (resource-based longest job first which allows execution of OpenCL jobs on any potential device including accelerators and CPU).

As has been discussed in the literature [25], the appropriate scheduling in terms of energy requires the voltage/frequency adjustment to be the main objective. While we do not have the voltage/frequency level as a direct parameter in the scheduling, it is included indirectly by the applied heuristics. More specifically, the efficient distribution of tasks onto multiple nodes and multiple processing elements on these nodes enables voltage scaling to be more effective. Depending on the execution scenario some of the processing elements can reduce the frequency and conserve

energy. Energy can also be considered as a primary objective but this requires a more sophisticated scheduling policy which may add non-tolerable delays.

These scheduling algorithms all have a common structure, where target CN selection is a two-step process: (1) queue assignment and (2) job selection. These are given in the next sections with more detail.

3.2.1 Selection of compute nodes based on resource requirements

In order to decide which nodes are optimal for satisfying a job's resource requirements, we consider the amount and type of each resource in the CN and the required resource amount/type. These resource requirements could be related to processing power, memory size, disk size, energy budget, communication bandwidth, etc. According to the job type, there could be more specific requirements. For example, an accelerator job may require to have a certain number of symmetric multi-processing (SMs) units, a certain texture size, a global memory size, and certain disk sizes. However, in this work, we currently use only the number of cores and SMs as resource requirements. Figure 4 gives the high-level view of the scheduling mechanism used in the proposed approach. As can be seen from this figure, the scheduling decision takes multiple parameters as input including the resource model, quality-of-service (QoS) model, the underlying infrastructure model, and the execution model.

3.2.2 Queuing jobs

We have two common job lists in order to keep track of submitted jobs, namely two groups in terms of job type. Although it would be possible to consider other types of jobs based on different criteria, our system currently considers two possibilities, CPU and accelerator jobs. In addition to the two common queues, we have also added a queue for handling OpenCL jobs. As mentioned before, our job assignment scheme has two steps. However, for RB-FIFO, RB-SJF, and RB-LJF most of the work is done in the queuing step. While queuing, our scheme considers profiling

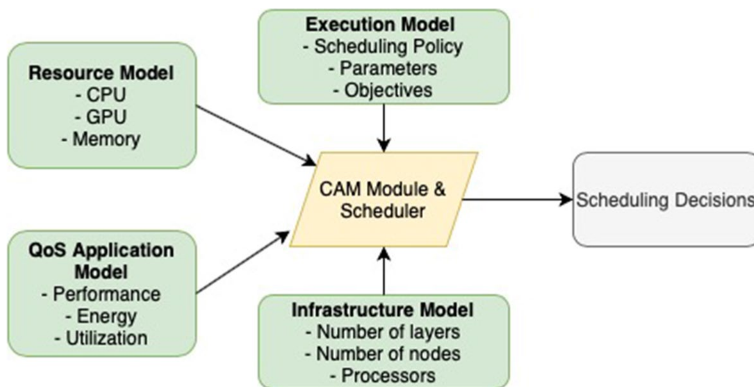


Fig. 4 High-level view of the scheduling mechanism used in the proposed approach

based execution time estimates and orders the priority of the submitted jobs in the corresponding queue. This way both energy and throughput can be improved since cumulative execution time directly affects the total energy consumption.

More specifically,

- For RB-FIFO, we keep a FIFO queue according to the arrival time of the jobs.
- For RB-SJF and RB-LJF, we order the jobs according to expected execution run time in ascending and descending order for RB-SJF and RB-LJF, respectively.
- For RB-LJF-BA and RB-LJF-OCL, again, we have an ordered queue but this time we also consider the aging of the job. More specifically, the value that we use to order the jobs is the sum of waiting time and expected execution time. Additionally, in RB-LJF-OCL, an OCL queue is added.

3.2.3 Selection of compute nodes

We developed two new heuristics for scheduling the jobs. The first heuristic is a resource-based longest job first with bottleneck avoidance. This heuristic avoids bottleneck cases for a node in the system by relaxing resource requests of some of the jobs. It also avoids starvation for a job by considering the aging of that job. The second heuristic is an extended version of the first one. This extended version enables executing some of the accelerator jobs in CPU for getting better throughput in addition to the resource-based longest job first scheme. For three basic approaches; RB-FIFO, RB-SJF, and RB-LJF; most of the work is done at queuing step. While scheduling, we start to traverse the corresponding queue from the beginning and find the first job that can execute on a free compute node based on resource requests from the user. In these basic approaches, we use cpu_jobs and acc_jobs queues, which are common for all heuristics. Algorithms 1 and 2 give the first two steps that are common for all baseline and developed scheduling heuristics.

Algorithm 1 Scheduling - CPU Job Selection

```
procedure SCHEDULE(cpu_jobs : list, acc_jobs : list, ocl_jobs : list)
    scheduled_jobs:JobList
    scheduled_nodes:NodeList
    for all Node n in system do
        if n 's CPU is available then
            for i = 0 to cpu_job.size()-1 do
                j ← cpu_job.get(i)
                if n is an optimal node for j then
                    selected_job ← j
                    cpu_jobs.remove(j)
                    scheduled_jobs.push_back(j)
                    scheduled_nodes.push_back(n)
                    cpu_is_optimal_exist ← TRUE
                    break
                end if
            end for
        end if
```

Algorithm 2 Scheduling - GPU Job Selection

```
if n 's GPU is available then
    for i = 0 to gpu_job.size()-1 do
        j ← pgu_job.get(i)
        if n is an optimal node for j then
            selected_job ← j
            acc_jobs.remove(j)
            scheduled_jobs.push_back(j)
            scheduled_nodes.push_back(n)
            gpu_is_optimal_exist ← TRUE
            break
        end if
    end for
end if
```

RB-LJF with bottleneck avoidance and RB-LJF-OCL scheduling heuristics use the same job selection logic. However, the RB-LJF-OCL heuristic is specialized to handle OpenCL jobs. As the first step, CAM tries to find CPU and/or accelerator jobs for the available node, where the highest priority lists are cpu_jobs and acc_jobs lists as in three basic heuristics. The algorithm given in Fig. 5 explains the details of our approach. As indicated before, this scheduling is only executed in RB-LJF-OCL. More specifically, CAM tries to find an accelerator job (OpenCL applications for our system) which also has a CPU version of the job to execute on an available CPU node only if CAM fails to find a CPU job in the first step. When selecting this job, CAM tries to select a job with lower CPU execution time than the first optimal accelerator job by

```
procedure SCHEDULE_OCL(cpu_jobs : list, acc_jobs : list, ocl_jobs : list)    ▷ OCL job
selection step
    if !cpu_job_found &
                        n's CPU is available then
        for i =ocl_job.size()-1 to 0 do
            j ← ocl_job.get(i)
            if n is an optimal node for j &
                        (j.CPUExecTime()≤first optimal
                        accelerator job's execution time) then
                selected_job:Job
                selected_job ← j
                ocl_job.remove(j)
                gpu_job.remove(j)
                scheduled_jobs.push_back(j)
                scheduled_nodes.push_back(n)
                cpu_job_found ← TRUE
                break
            end if
        end for
    end if
end procedure
```

Fig. 5 Scheduling—OCL Job Selection to Execute on CPU

traversing the queue in descending order in terms of execution times (ordering of the queue is LJF). This way, we are able to prevent accelerator jobs to be executed on CPU which can block a CN's CPU for a long time.

If CAM fails to find a job for CPU and accelerator in the first three steps, it tries to find a job with the smallest execution time from cpu_jobs and acc_jobs lists by applying a decay function. In order to enable this, CAM searches cpu_jobs and acc_jobs list starting from the shortest job and finds a matching job according to the selected decay function. For the decay function, we have used the number of cores (for CPU) and the number of SMs (for GPU) as resource types. Every time that the scheduler is called and there are no jobs for the specific node, we have increased the number of free passes for that specific node. Then, during scheduling time, we have relaxed resource requests of the jobs by 0.5^{free_passes} while selecting a job for the idle node. Jobs are executed on a node for the entire duration of the execution. That is, jobs do not migrate between nodes to prevent potential data migration overheads.

4 Integer linear programming (ILP) formulation

Our goal in this section is to present an ILP formulation of the problem of scheduling jobs within the given framework. This ILP-based management scheme optimally decides on job scheduling, thereby providing an upper-bound for our heuristic implementations.

ILP is a set of techniques that solve optimization problems where both the objective function and the constraints are linear functions. The solution variables generated as the result are restricted to be integers. A special case of ILP is Binary Integer Programming (BIP or 0-1 ILP) where variables are required to be 0 or 1 (rather than arbitrary integers). It is used in this ILP formulation for determining the application-to-processor or application-to-accelerator mappings. While ILP uses the same inputs as the heuristic approaches, it generates the best possible mapping based on the given parameters.

In order to test our approach, we use a commercial tool, XPressMP [47] to solve our ILP problem. Below, we give the important constant terms and decision variables used in our ILP formulation.

4.1 Constants

In the ILP formulation, we assume that there is a fixed number of jobs (J) and a fixed number of nodes (N) in the system.

There are also job related constants given as input to the ILP solver. $EXEC_{i,j}$ indicates the execution time of job i on processor/accelerator j, whereas $SUBMIT_j$ indicates the submit time of job j.

4.2 Variables

We capture the assignment of a certain job on specific node using the $assign_{i,j}$ variable, where job j is executed on resource i. Note that, resource in this case could be a

processor or an accelerator on a node. Another binary variable used is $colocated_{i,j,k}$, where job j and job k are executed on the same resource i. $precedes_{j,k}$ is used to indicate if job j precedes job k in execution. To capture the start and end time of a job, two new 0-1 variables are used, namely $start_j$ and $finish_j$, respectively. To capture when the execution of all jobs have completed, we use $busy_i$, where i indicates an execution point in time.

4.3 Constraints

Our first constraint is about executing the jobs on only one resource. Therefore, we need to have the following constraint to capture this.

$$\sum_{i=1}^{N} assign_{i,j} = 1, \quad \forall j. \tag{1}$$

Following constraint determines whether two different jobs are executed on the same resource or not. If both $assign_{i,j}$ and $assign_{i,k}$ is "1" (job j and job k are executed on resource i) then $colocated_{j,k}$ is forced to be "1".

$$colocated_{j,k} \geq assign_{i,j} + assign_{i,k} - 1, \quad \forall i, j, k. \tag{2}$$

If both job j and job k are executed on the same resource then one of the jobs must precede the other one. Either $precedes_{j,k}$ or $precedes_{k,j}$ must be "1".

$$precedes_{j,k} + precedes_{k,j} = colocated_{j,k}, \quad \forall i, j, k. \tag{3}$$

If job k precedes job j then there must be a difference between start times of two jobs at least as much as the execution time of job k. Note that, constant M in the following expression is assumed to be close to ∞.

$$start_j - start_k \geq \sum_{i=1}^{N} assign_{i,k} \times EXEC_{i,k} - M \times (1 - precedes_{k,j}), \quad \forall j, k. \tag{4}$$

Following constraint forces each job's start time to be greater than its submission time.

$$start_j \geq SUBMIT_j, \forall j. \tag{5}$$

Similarly, a job's finish time is the sum of its start time and the total execution time on the assigned node. Therefore,

$$finish_j = start_j + \sum_{i=1}^{N} assign_{i,j} \times EXEC_{i,j}, \quad \forall j. \tag{6}$$

4.4 Objective function

In the following constraint, *busy* corresponds to finish time of the last job which is executed in the system. In this constraint, *busy* is forced to be greater than finish times of all jobs. This way we can capture the finish times of all the jobs. This is included in the objective function since it is the basis for our objective.

$$busy_j \geq finish_j, \forall j. \tag{7}$$

Then, using these busy binary variables we can decide if the jobs are successfully finished or not. Therefore, our formal objective function is to minimize the execution time of a set of applications using this *busy* variable.

$$minimize\left(\sum_{j=1}^{N} busy_j\right). \tag{8}$$

Note that, in our ILP formulation, we employ execution time as the main constraint, whereas energy, storage, communication bandwidth, and other possible constraints are left out. For example, depending on the switch present in a node, bandwidth available to the connected links will be limited. Our ILP formulation, in its current form, does not cover these possible constraints.

However, our formulation can easily be modified to include such constraints. In addition to additional constraints, our ILP formulation can also be modified to optimize for a different objective function such as energy instead of execution time.

5 Experimental evaluation

In order to compare our approach with the state-of-the-art implementations, we developed heuristics; namely, Resource-Based First In First Out (RB-FIFO), Resource-Based Longest Job First (RB-LJF), and Resource-Based Shortest Job First (RB-SJF). Moreover, we implemented an optimal mapping approach where an integer linear programming (ILP) optimization is used for comparison.

5.1 Setup

We tried to simulate a heterogeneous cluster by using different nodes with different hardware components and properties. Specific properties of nodes we have used are given in Tables 1 and 2.

5.2 Benchmarks

To test our system, we have selected 28 different OpenCL jobs from well known parallel benchmarks [28]. Details of the benchmarks we used in the experiments are given in Table 3. Each benchmark has different characteristics; some have over 50 kernels and others have only few kernels. We used different problem sizes (i.e. S and

Table 1 24 Compute nodes used in system setup 1

	CPU	GPU	GPU
Node 1–8	6 cores AMD Phenom II	Nvidia GeForce GTX 460	Nvidia Tesla K20
Node 9–16	12 cores Intel Xeon	Nvidia Tesla K20	Nvidia Tesla K40
Node 17–24	16 cores Intel Xeon	Nvidia Tesla K40	Nvidia GeForce GTX 460

Table 2 16 Compute nodes used in system setup 2

	CPU	GPU	GPU
Node 1–8	6 cores AMD Phenom II	Nvidia GeForce GTX 460	Nvidia Tesla K20
Node 9–16	12 cores Intel Xeon	Nvidia Tesla K20	Nvidia Tesla K40

W classes of NAS benchmarks) to determine the effect of data size on kernel mapping. As evident in Table 3, the tendency of kernels may change with different problem sizes, which is basically due to the characteristics of that particular kernel. For example, it is better to run benchmark SP with W class data set on only CPU, while it is not the case for the same benchmark with S class data set.

To test such applications, we used AMD-APP SDK [2] for compiling OpenCL applications for X86 ISA. These benchmarks are used to create 2 groups of workloads. In the first group (*group A*), 3 different versions of 28 different jobs are used (in total 76 jobs). Different versions of jobs are created by multiple executions of each job (1x, 2x, 3x). In order to facilitate the execution of the simulation, we implemented 4 different test batches. Submission times of jobs are determined by a random number generator which produces a sequence of integers by using a Gaussian distribution. These randomly generated times are sorted in ascending order. The first group of workloads (Workload 1, 2, 3, and 4) is tested with 2 different system setups.

In the first group of executions, resource requests for jobs are set to the node with the least resources. The first group of workloads is used for showing the efficiency of the system in terms of execution time and shows the effect of OCL extension on different heterogeneous environments and job submission scenarios. In the second group (*group B*), 28 jobs are selected and used. Similar to the first group, 4 different test cases are generated and job submissions are simulated. The second group of workloads' (Workload 5–8) resource requests for jobs are set to the node with the highest resources in order to create contention on a specific node. We use this second group of workloads to show the effect of the RB-LJF-BA heuristic.

5.3 Experimental results

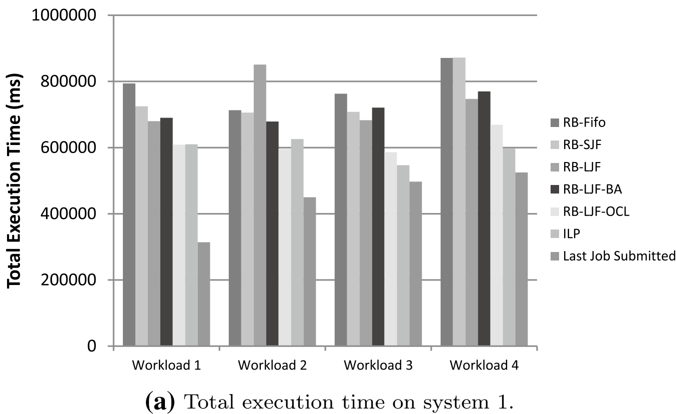
We experimented with our system with different configurations and job submission scenarios. To compare different scheduling approaches, we have implemented and tested RB-FIFO, RB-SJF, RB-LJF, RB-LJF-BA, and RB-LJF-OCL. In order to

Table 3 The descriptions and problem sizes of benchmarks used in our experiments [28]

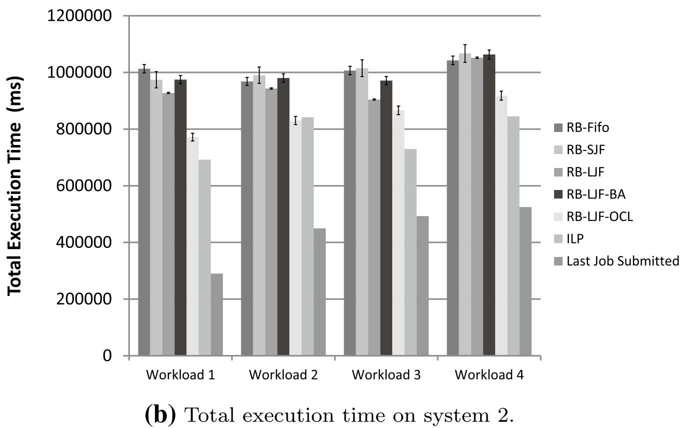
Benchmark name	Description	Parameters	Class S	Class W
BT	Solves multiple, independent systems of non-diagonally dominant, block-tridiagonal equations	grid size	$12 \times 12 \times 12$	$24 \times 24 \times 24$
		No. of iterations	60	200
		Time step	0.01	0.0008
CG	Computes an approximation to the smallest eigenvalue of a large, sparse, symmetrically positive definite matrix using a conjugate gradient method	No. of rows	1400	7000
		No. of nonzeros	7	8
		No. of iterations	15	15
		Eigenvalue shift	10	12
EP	Evaluates an integral by means of pseudo random trials	No. of random-number pairs	2^{24}	2^{25}
LU	A regular-sparse, block (5×5) lower and upper triangular system solution	Grid size	$12 \times 12 \times 12$	$33 \times 33 \times 33$
		No. of iterations	50	300
		Time step	0.5	0.0015
SP	Solves multiple, independent systems of non-diagonally dominant, scalar, pentadiagonal equations	Grid size	$12 \times 12 \times 12$	$36 \times 36 \times 36$
		No. of iterations	100	400
		Time step	0.015	0.0015

show the effectiveness of these scheduling policies, we have also implemented and tested an optimal scheduling policy through ILP. Figure 6a shows the base results for different workloads running on system setup 1. Last job submitted indicates a minimum bound for completing all the jobs in the system. We can observe from this figure that the run-time of the system with RB-LJF-BA heuristic gives an average execution time between maximum and minimum runtimes of systems with RB-FIFO, RB-SJF, and RB-LJF heuristics. However, a critical gain for this heuristic is seen when user requests create a bottleneck on some of the nodes.

In the next set of experiments, we evaluate the execution time reduction by using OCL extension over the best case for different workloads on system setup 1. As can be seen from Fig. 7a, by using OCL extension which allows us to port some accelerator applications to CPU, we have an average 12% execution time reduction in the first group of tests. Meanwhile, RB-LJF-OCL execution time is close to optimal execution time. In some of the ILP experiments, due to a large number of variables and constraints, ILP solver [46] was not able to find the final

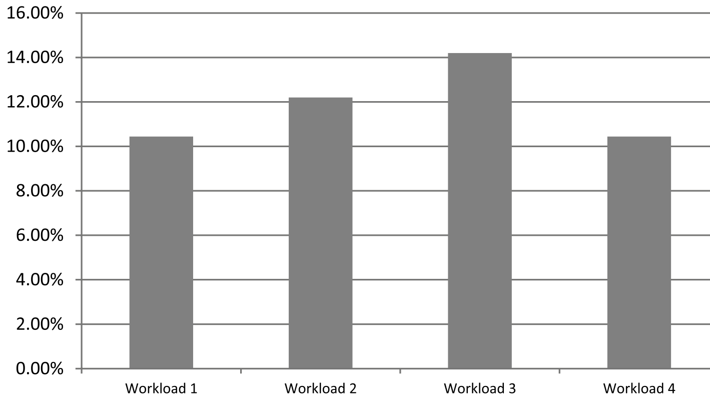


(a) Total execution time on system 1.

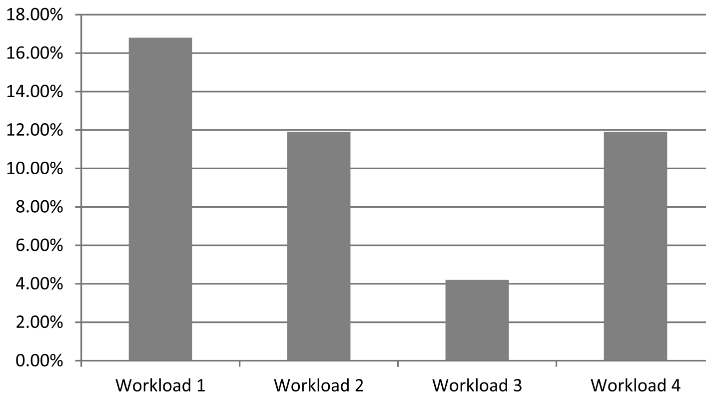


(b) Total execution time on system 2.

Fig. 6 Total execution time comparison with group A workloads



(a) Average execution time reduction on system 1 when compared with respect to results given in Figure 6a.



(b) Average execution time reduction on system 2 when compared with respect to results given in Figure 6b.

Fig. 7 Execution time reduction by using OCL extension

solution. In such cases, we have used the best bound found by the solver. Since this happened occasionally, its impact is negligible. Overall, the maximum gap between the best bound and best solution found was 15%. Similarly, the maximum difference between two runs of the same scheduling heuristic with the same load was found to be 7.2%. These are all within acceptable and tolerable ranges when the similar implementations are considered. Note that, while ILP is an optimal approach that can be used for the decisions, it incurs considerable execution time overheads due to a large number of variables and constraints with even a small-sized problem. This is even pronounced when the online nature of the decisions is considered.

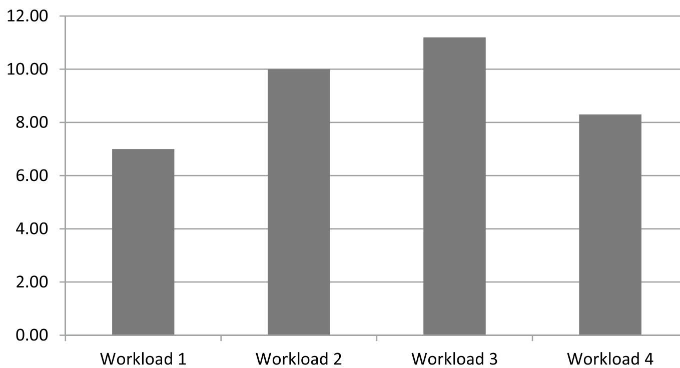
In the second group of tests, we evaluate the effect of the RB-LJF-BA heuristic. Specifically, Fig. 6b shows the execution time results for different workloads

on system 2. We observe that the execution latencies of RB-LJF-OCL and minimum execution latencies for different workloads are close.

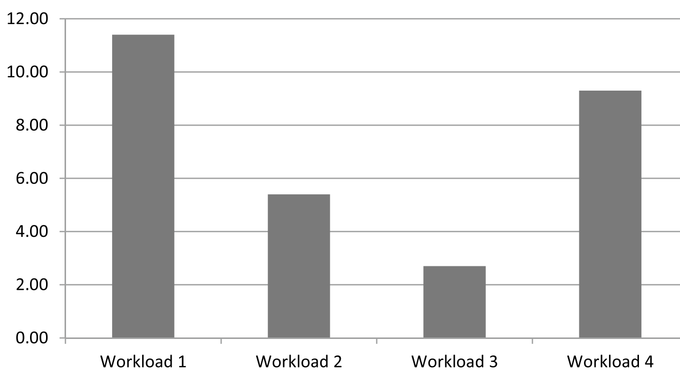
A similar observation can be made in Fig. 7b, where execution time reduction when using OCL extension over the best case for different workloads is shown. In these experiments, the maximum difference between two runs of the same scheduling heuristic with the same workload is found to be 3.4%. For these tests, we were able to find the minimum run times for each workload.

Figure 8a, b show the energy reductions with OCL extension over the best case for different workloads for System 1 and System 2, respectively. In these experiments, the maximum difference between two runs of the same scheduling heuristic with the same workload is found to be 4.1%.

In the last set of experiments, we measure the effect of the bottleneck avoidance approach. Specific results are shown in Fig. 9a, b. As can be seen from these figures, bottleneck avoidance improves system execution time when compared

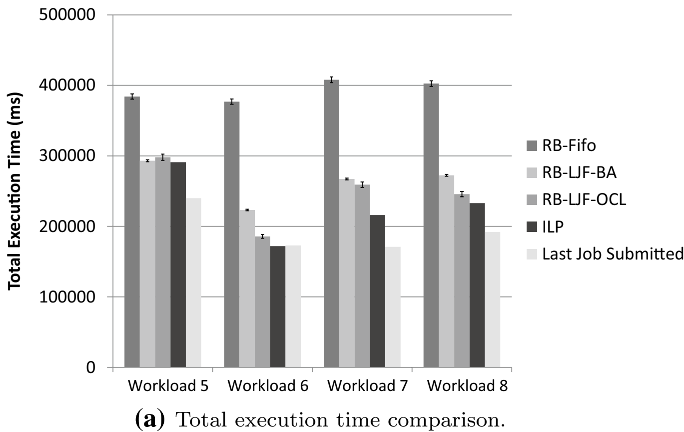


(a) Average energy reduction on system 1.



(b) Average energy reduction on system 2.

Fig. 8 Energy reduction by using OCL extension



(a) Total execution time comparison.

(b) Execution time reduction by bottleneck avoidance (BA) and OCL extension.

Fig. 9 Experiments with group B workloads on system 2

with RB-FIFO. More specifically, we can reduce up to 50% execution time by relaxing user requests for jobs.

6 Conclusion

Currently, there are management schemes for conventional clusters and cloud systems ranging from the operating system (OS) level to application level. Our accelerator-aware computing system management framework, on the other hand, enables the users to utilize the underlying architecture in the most effective way. Specifically, we implement a runtime system where users can submit jobs which will then be assigned to a corresponding node or accelerator depending on the application features such as execution time and data requirements. We implemented the computing management system in an in-house testbed, where heuristic-based management

algorithms are tested. These heuristics consider application characteristics and respective execution properties on the nodes and accelerators. In addition, we implemented an ILP-based management scheme which optimally decides on application mapping, thereby providing an upper-bound for our heuristic implementations. Our experimental evaluations show that including accelerator support in the cluster management framework improves system execution time significantly. We believe that this approach has the potential to provide an effective solution for next-generation accelerator-based heterogeneous computing systems.

Acknowledgement This work has been supported in part by a grant from Turkish Academy of Sciences and a grant from Türk Telekom (Project Number: 3015-04).

References

1. Alma and the European Alma Regional Centre. http://www.eso.org/sci/facilities/alma.html. Accessed 07 May 2021
2. Amd app sdk. http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/. Accessed 07 May 2021
3. Amd Radeon HD 6990. http://www.amd.com/us/products/desktop/graphics/amd-radeon-hd-6000/hd-6990/Pages/amd-radeon-hd-6990-overview.aspx. Accessed 07 May 2021
4. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. Commun. ACM 53(4):50–58. https://doi.org/10.1145/1721654.1721672
5. Bao Z, Chen C, Zhang W (2018) Task scheduling of data-parallel applications on HSA platform. In: Zhou Q, Gan Y, Jing W, Song X, Wang Y, Lu Z (eds) Data Science. Springer, Singapore, pp 452–461
6. Barik R, Farooqui N, Lewis BT, Hu C, Shpeisman T (2016) A black-box approach to energy-aware scheduling on integrated cpu-gpu systems. In: 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp 70–81
7. Bellavista P, Corradi A, Reale A, Ticca N (2014) Priority-based resource scheduling in distributed stream processing systems for big data applications. In: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp 363–370. https://doi.org/10.1109/UCC.2014.46
8. Chen YR, Lee CR (2016) G-storm: a gpu-aware storm scheduler. In: 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp 738–745. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.130
9. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun. of ACM 51(1):107–113
10. Dev K, Zhan X, Reda S (2016) Power-aware characterization and mapping of workloads on cpu-gpu processors. In: 2016 IEEE International Symposium on Workload Characterization (IISWC), pp 1–2. https://doi.org/10.1109/IISWC.2016.7581285
11. Doka K, Papailiou N, Tsoumakos D, Mantas C, Koziris N (2015) Ires: intelligent, multi-engine resource scheduler for big data analytics workflows. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15. ACM New York, pp 1451–1456. https://doi.org/10.1145/2723372.2735377
12. Du P, Sun Z, Zhang H, Ma H (2019) Feature-aware task scheduling on CPU-FPGA heterogeneous platforms. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp 534–541

13. Fang J, Zhang J, Lu S, Zhao H (2020) Exploration on task scheduling strategy for cpu-gpu heterogeneous computing system. In: 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp 306–311
14. Feitelson DG, Rudolph L, Schwiegelshohn U (2005) Parallel job scheduling—a status report. In: Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing, JSSPP'04. Springer, Berlin, Heidelberg, pp 1–16. https://doi.org/10.1007/11407522_1
15. Goswami A, Young J, Schwan K, Farooqu N, Gavrilovska A, Wolf M, Eisenhauer G (2016) Gpushare: fair-sharing middleware for gpu clouds. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp 1769–1776. https://doi.org/10.1109/IPDPSW.2016.94
16. GPU based cloud computing. http://www.ogf.org/OGF28/materials/1914/OpenGridForum28.pdf. Accessed 07 May 2021
17. GPU based cloud computing—open grid forum. http://www.ogf.org/OGF28/materials/1914/OpenGridForum28.pdf. Accessed 07 May 2021
18. Graphics processing unit. http://en.wikipedia.org/wiki/Graphics_processing_unit. Accessed 07 May 2021
19. Graham RL, Shipman GM, Barrett BW, Castain RH, Bosilca G, Lumsdaine A (2006) Open MPI: a high-performance, heterogeneous MPI. In: 2006 IEEE International Conference on Cluster Computing, pp 1–9. https://doi.org/10.1109/CLUSTR.2006.311904
20. IBM, Zikopoulos P, Eaton C (2011) Understanding big data: analytics for enterprise class hadoop and streaming data, 1st edn. McGraw-Hill Osborne Media
21. Intel many integrated core architecture. http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html. Accessed 07 May 2021
22. Lee G, Chun BG, Katz H (2011) Heterogeneity-aware resource allocation and scheduling in the cloud. In: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'11. USENIX Association, Berkeley, p 4. http://dl.acm.org/citation.cfm?id=2170444.2170448
23. Liu X, Liu P, Hu L, Zou C, Cheng Z (2020) Energy-aware task scheduling with time constraint for heterogeneous cloud datacenters. Concurr Comput: Pract Exp 32(18):e5437. https://doi.org/10.1002/cpe.5437 E5437 cpe.5437
24. Ma S, Jiang J, Li B, Li B (2016) Custody: towards data-aware resource sharing in cloud-based big data processing. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp 451–460. https://doi.org/10.1109/CLUSTER.2016.59
25. Mei X, Chu X, Liu H, Leung YW, Li Z (2017) Energy efficient real-time task scheduling on cpu-gpu hybrid clusters. In: IEEE INFOCOM 2017—IEEE Conference on Computer Communications, pp 1–9. https://doi.org/10.1109/INFOCOM.2017.8057205
26. Memeti S, Li L, Pllana S, Kolodziej J, Kessler C (2017) Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption. In: Proceedings of the 2017 workshop on adaptive resource management and scheduling for cloud computing (ARMS-CC '17). Association for Computing Machinery, New York, NY, USA, pp 1–6. https://doi.org/10.1145/3110355.3110356
27. Mohammadi R, Shekofteh SK, Naghibzadeh M, Noori H (2016) A dynamic special-purpose scheduler for concurrent kernels on gpu. In: 2016 6th International Conference on Computer and Knowledge Engineering (ICCKE), pp 218–222. https://doi.org/10.1109/ICCKE.2016.7802143
28. Nas benchmark. http://www.nas.nasa.gov/publications/npb.html. Accessed 03 May 2021
29. Nvidia's next generation cuda compute architecture: Fermi. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf. Accessed 07 May 2021
30. Pinto VG, Stanisic L, Legrand A, Schnorr LM, Thibault S, Danjean V (2016) Analyzing dynamic task-based applications on hybrid platforms: an agile scripting approach. In: 2016 3rd Workshop on Visual Performance Analysis (VPA), pp 17–24. https://doi.org/10.1109/VPA.2016.008
31. Open mpi. https://www.open-mpi.org/faq/?category=running. Accessed 07 May 2021
32. Opencl project. http://www.khronos.org/opencl/. Accessed Accessed 03 May 2021
33. Openmp. http://openmp.org/. Accessed 07 May 2021
34. Ravi V, Becchi M, Agrawal G, Chakradhar S (2012) Valuepack: value-based scheduling framework for CPU-GPU clusters. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, pp. 1–12. https://doi.org/10.1109/SC.2012.111

35. Ravi VT, Becchi M, Jiang W, Agrawal G, Chakradhar S (2012) Scheduling concurrent applications on a cluster of CPU-GPU nodes. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012), CCGRID '12. IEEE Computer Society, Washington, pp 140–147. https://doi.org/10.1109/CCGrid.2012.78
36. Sabin G, Kettimuthu R, Rajan A (2003) Scheduling of parallel jobs in a heterogeneous multi-site environment. In: The Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, pp 87–104
37. Sabin G, Sahasrabudhe V, Sadayappan P (2005) Assessment and enhancement of meta-schedulers for multi-site job sharing. In: High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on, pp 144–153. https://doi.org/10.1109/HPDC.2005.1520949
38. Scogland TR, Rountree B, Feng Wc, de Supinski BR (2012) Heterogeneous task scheduling for accelerated openmp. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium, pp 144–155. https://doi.org/10.1109/IPDPS.2012.23
39. Shirahata K, Sato H, Matsuoka S (2010) Hybrid map task scheduling for GPU-based heterogeneous clusters. In: Cloud Computing Technology and Science (CloudCom), 2010 IEEE 2nd International Conference on, pp 733–740. https://doi.org/10.1109/CloudCom.2010.55
40. Shulga DA, Kapustin AA, Kozlov AA, Kozyrev AA, Rovnyagin MM (2016) The scheduling based on machine learning for heterogeneous CPU/GPU systems. In: 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), pp 345–348. https://doi.org/10.1109/EIConRusNW.2016.7448189
41. The nist definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Accessed 07 May 2021
42. Tegra mobile processors. http://www.nvidia.com/object/tegra.html. Accessed 07 May 2021
43. Torque. http://www.adaptivecomputing.com/products/open-source/torque/. Accessed 07 May 2021
44. Ukidave Y, Li X, Kaeli D (2016) Mystic: predictive scheduling for gpu based cloud servers using machine learning. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 353–362. https://doi.org/10.1109/IPDPS.2016.73
45. Wan L, Zheng W, Yuan X (2021) Efficient inter-device task scheduling schemes for multi-device co-processing of data-parallel kernels on heterogeneous systems. IEEE Access 9:59968–59978
46. Xpress mp. http://decisions.fico.com/aboutXpress.html. Accessed 07 May 2021
47. Xpress-mp optimizer reference manual, fico®xpress optimization. https://www.fico.com/en/products/fico-xpress-optimization. Accessed 07 May 2021
48. Young J, Shon SH, Yalamanchili S, Merritt A, Schwan K, Fröning H (2013) Oncilla: a gas runtime for efficient resource allocation and data movement in accelerated clusters. In: 2013 IEEE International Conference on Cluster Computing (CLUSTER), pp 1–8. https://doi.org/10.1109/CLUSTER.2013.6702679
49. Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I (2008) Improving mapreduce performance in heterogeneous environments. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08. USENIX Association, Berkeley, pp 29–42. http://dl.acm.org/citation.cfm?id=1855741.1855744
50. Zhang K, Wu B (2012) Task scheduling for gpu heterogeneous cluster. In: 2012 IEEE International Conference on Cluster Computing Workshops, pp 161–169. https://doi.org/10.1109/ClusterW.2012.20
51. Zhang L, Wu C, Li Z, Guo C, Chen M, Lau F (2013) Moving big data to the cloud: an online cost-minimizing approach. Sel Areas Commun, IEEE J 31(12):2710–2721. https://doi.org/10.1109/JSAC.2013.131211

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.