Published in final edited form as:
Proc Symp Comput Archit High Perform Comput. 2014 October ; 2014: 89–96. doi:10.1109/SBAC-PAD.2014.15.

Efficient Execution of Microscopy Image Analysis on CPU, GPU, and MIC Equipped Cluster Systems

G. Andrade,
Federal University of Minas Gerais

R. Ferreira,
Federal University of Minas Gerais

George Teodoro,
University of Brasília

Leonardo Rocha,
Federal University of São João del Rei

Joel H. Saltz, and
Stony Brook University

Tahsin Kurc
Stony Brook University

G. Andrade: gnandrade@dcc.ufmg.br; R. Ferreira: renato@dcc.ufmg.br; George Teodoro: teodoro@cic.unb.br; Leonardo Rocha: lcrocha@ufsj.edu.br; Joel H. Saltz: joel.saltz@stonybrookmedicine.edu; Tahsin Kurc: tahsin.kurc@stonybrook.edu

Abstract

High performance computing is experiencing a major paradigm shift with the introduction of accelerators, such as graphics processing units (GPUs) and Intel Xeon Phi (MIC). These processors have made available a tremendous computing power at low cost, and are transforming machines into hybrid systems equipped with CPUs and accelerators. Although these systems can deliver a very high peak performance, making full use of its resources in real-world applications is a complex problem. Most current applications deployed to these machines are still being executed in a single processor, leaving other devices underutilized. In this paper we explore a scenario in which applications are composed of hierarchical data flow tasks which are allocated to nodes of a distributed memory machine in coarse-grain, but each of them may be composed of several finer-grain tasks which can be allocated to different devices within the node. We propose and implement novel performance aware scheduling techniques that can be used to allocate tasks to devices. We evaluate our techniques using a pathology image analysis application used to investigate brain cancer morphology, and our experimental evaluation shows that the proposed scheduling strategies significantly outperforms other efficient scheduling techniques, such as Heterogeneous Earliest Finish Time - HEFT, in cooperative executions using CPUs, GPUs, and MICs. We also experimentally show that our strategies are less sensitive to inaccuracy in the scheduling input data and that the performance gains are maintained as the application scales.

I. Introduction

A novel application scenario that has been arising in the past few years, termed Data Science, consists of extracting meaningful knowledge from large datasets collected and stored in many different areas of science and engineering. In this scenario, applications consist of several stages of data extraction and transformation, by a large variety of complex algorithms, which, associated to the large volumes of the raw data, impose a high demand for computing power.

Image analysis is one of the applications in this context, for instance in microscopy imaging studies like pathology image analysis used to investigate brain cancer morphology. In this case, datasets of images with 120K×120K pixels are processed in many steps, such as normalization, segmentation, feature computation and classification, in order to derive some insight from the raw data. In a study, several executions of the processing are necessary, with different parameters and algorithms, in order to explore a variety of aspects of the data and the process. However, in order to achieve reasonable execution time for the study to be feasible, significant computing power need to harvested.

Concomitantly, a novel trend in computer systems' architecture is the utilization of large clusters of nodes with high-end processors associated with compute-intensive, massively parallel co-processors (www.top500.org). Two of the most popular co-processors are the GPUs and Intel Xeon Phi. These co-processors have emerged as alternative architectures as the golden standard of frequency scaling broke down in the first decade of the century. It consists of massively parallel architectures privileging ALU operations over I/O and control flow operations. The end result is that the co-processors can yield very high compute density if certain criteria is matched by the application and its data.

The hiatus between the complexity of the data science algorithms and that of the current high performance platforms has given a new boost to computer system research and several novel frameworks [1], [2], [3], [4], [5], [6], [7], [8] have been proposed to bridge the gap, providing high level programming abstractions for algorithm design while allowing efficient execution on current hardware. One such system is Extreme DataCutter [9]. In this framework applications are represented by hierarchical data flows and each stage of the application may be replicated and assigned for computation in several nodes of a hybrid distributed memory machine.

The focus of this paper is to explore the cooperative execution of a pathology image analysis application used to investigate brain cancer morphology on hybrid systems equipped with CPUs, GPUs, and MICs. More specifically we propose, implement and evaluate performance aware scheduling techniques, which take into account that a given device is more efficient for specific tasks, to maximize the utilization of hybrid systems. Consequently, we make the pathology image analysis application computational feasible.

We evaluate the performance of our proposed schedulers against traditional scheduling strategies and show that they significantly outperforms the traditional ones, such as Heterogeneous Earliest Finish Time (HEFT), in cooperative executions using CPUs, GPUs, and MICs. We also experimentally show that our strategies are less sensitive to inaccuracy in the scheduling input data and that the performance gains are maintained as the application scales.

II. Motivating Application

This work is primarily motivated by our in silico studies of brain cancer [10], which intends to find better tumor classifications using complementary datasets of high resolution whole tissue slide images (WSIs), gene expression data, clinical data, and radiology images. WSIs may have several layers of 120K×120K pixels. The basic work flow in our application has the following stages: 1) preprocessing as color normalization,2) segmentation of cells and nuclei (objects), 3) extraction of a set of shape and texture features per object, and 4) classification based on object features. The most time consuming phases of the application are segmentation and features computation and, as such, they have been our focus for execution in large-scale hybrid machines.

The segmentation stage detects cells and nuclei and delineates their boundaries. It consists of several operations, forming a data flow graph (see Figure 1). The operations in the segmentation include morphological reconstruction to identify candidate objects, watershed segmentation to separate overlapping objects, and filtering to eliminate candidates that are unlikely to be nuclei. The feature computation stage derives quantitative attributes in the form of a feature vector for individual segmented objects. The feature types include pixel statistics, gradient statistics, edge detection, and morphometry. Most of the features can be computed independently.

In a previous work [11], [12], we have implemented and evaluated the performance of these operations on CPUs, GPUs, and on the Intel Xeon Phi (MIC). The operations were implemented on all devices using the same parallelization strategy and level of optimizations. CPU code was implemented in C++, which was annotated with OpenMP for execution on MIC. The GPU versions were implemented in CUDA. The speedups attained by the operations on accelerators, using a single CPU core execution as baseline, are presented in Figure 2.

We have classified these operations in three groups with similar computation and data access patterns: (1) Regular data access: RGB2Gray, Morphological Open, Color De-convolution, Pixel Stats, Gradient Stats, and Sobel Edge; (2) Irregular data access: Morphological Reconstruction, FillHoles, and Distance Transform; (3) Operations that rely on atomic instruction: Area Threshold and Connected Component Labeling (CCL). These characteristics along with computation intensity of operations were correlated with processor specifications and our own benchmarks to explain their performance. More detailed analysis can be found in our earlier work [11].

In this work, we are interested in providing efficient strategies for cooperative execution on hybrid systems, equipped with multiple accelerators. The ideas and strategies proposed in this paper are based on the following observations from Figure 2: *(i) there exists a high variability on the speedups achieved by the same processor as different operations are considered; (ii) the relative performance among processors (CPU, GPU, and MIC) varies according to the operation executed.* These facts suggest that different processors are more efficient for particular types of operations (or data access and computation patterns). Further, an efficient scheduling for cooperative execution on hybrid systems should take into account these performance variabilities to maximized the aggregate computing power of a heterogeneous machine.

III. Extreme DataCutter

The Extreme DataCutter (EDC) [9] is a runtime system for heterogeneous environment, which supports the execution of data flow applications. Such as in DataCutter [13], each stage of the application may be replicated and assigned to different nodes of a distributed memory machine. In EDC, applications may be expressed as a hierarchical data flow in which an application stage may be implemented as another data flow of finer-grain operations. EDC executes applications using a bag-of-tasks style for assignment of coarse-grain stage instances to nodes of the machine, which is combined with a task dependency resolution scheme to assert that stages are only assigned for execution after dependencies are resolved.

An Manager-Worker model is employed by the system to schedule and execute the application coarse-grain data flow, as presented in Figure 3. The user is responsible for developing a part of the Manager code that instantiates the coarse-grain stages and sets dependencies among them. Each node of the compute environment executes a single instance of the Worker process, and stage instances are assigned for computation to Workers in a demand-driven way until all application stage instances are executed. The Manager-Worker communication implemented in Message Passing Interface (MPI).

A Worker is able to use all the resources within a compute node (i.e., CPU cores, GPUs and MICs). Multiple stage instances may be assigned to the same Worker concurrently in order to fully utilize the devices, the maximum number of which is a parameter called Worker Request Window Size. As shown in Figure 4, a Worker is built from several components. The Worker Communication Controller (WCC) is responsible for communication with the Manager. The management of computing devices is executed with a Worker Resource Manager (WRM). Once a stage instance is received for execution, it may instantiate a data flow of finer-grain operations to carry out the stage computation. These operations/tasks are dispatched for computation by the WRM, which will assign tasks for execution to the available devices. The WRM creates a thread for each CPU core, GPU or MIC available. The device manager threads notify the WRM once they become available, and the WRM scheduler selects one of the tasks with dependencies resolved for execution. After all operations dispatched by a stage instance have finished their execution, WCC notifies the Manager which releases the dependencies of the completed stage instance.

IV. Cooperative Execution on Hybrid Systems

This section describes the deployment of our application on EDC and describes our proposed scheduling strategies.

A. Application Deployment on EDC

In order to create a high performance version of the application, we implemented multiple versions of the fine-grain operations within the segmentation and feature computation stages (Figure 1): C++ for CPUs, CUDA for GPUs an C++ with OpenMP annotations for the Intel Phi, as described in our earlier work [11]. The application was then deployed using EDC for parallel execution on a distributed memory machine. We used segmentation and feature computation as the first level, coarse-grain stages, each of which were implemented using finer-grain data flow tasks. The finer-grain tasks are then scheduled to the different devices within a compute node as per our proposed scheduling algorithms.

Our application also partitions the input images into multiple tiles that are independently analyzed by the coarse-grain stages, allowing separate instances of them to be created per image tile. Moreover, the entire application graph is only known at execution time as features are computed based upon the image segmentation. EDC’s dynamic computation graph composition capability comes in handy in this scenario, allocating compute resources as needed. The application was profiled beforehand to measure execution times of the individual operations, which were used to emulate the skeleton of the application and evaluate the proposed schedulers.

B. Performance Aware Scheduling

We now propose our scheduling strategies for efficient execution on heterogeneous systems with accelerators. As mentioned earlier, operations may achieve different benefits according to the platform used. Therefore, we propose performance-aware task scheduling techniques, which take into account performance variabilities to better utilize hybrid systems. The proposed strategies, described below, are implemented in Extreme DataCutter’s WRM module, and are used to schedule fine-grain tasks created within each stage of the application to CPUs, GPUs, and/or MICs.

Performance-Aware Dequeue Adapted Scheduler (PADAS): is the first strategy we propose. It is implemented using a single queue that stores tasks sorted according to their expected speedup on the accelerators. If the expected speedup on a MIC is higher than that on a GPU, the task is inserted in a sorted order starting at head of the queue, and it is placed just before a task with smaller expected speedup on a MIC is found. A similar approach is used when speedup on a GPU is higher than on a MIC. In that case, the insertion starts in the other end of the queue (tail), and the task is placed just before a task with smaller speedup on a GPU. This strategy intends to use a single queue in which tasks more appropriate for a MIC, for instance, would be in one end of the queue, whereas tasks with highest speedups for GPUs would be in the other end. Similarly to other scheduling strategies implemented in this work, the assignment of tasks to devices is carried out on a demand-driven basis as the thread managing a computing device becomes available. If the available device is a MIC, the task first task in the head of the queue is selected, and the task in the tail is chosen in the case of a GPU. When the available processor is a CPU core, the a task in the middle of the queue is selected for execution. The actual point in the middle of queue refers to the limit between tasks inserted by MIC and GPU (see Figure 5). This approach aims to assign tasks inefficiently computed by accelerators to CPU cores.

Performance-Aware Multiqueue Scheduler (PAMS): is a variation of the first strategy in which sorted queue is created per type of computing device used. For example, if CPU, GPU, and MIC are used cooperatively in an execution, three queues are instantiated. Further, every task is inserted in all queues, which are sorted in a decreasing order according to the expected speedup for that device. The CPU queue is sorted according to the maximum speedup value of the tasks in other devices, i.e., *max*(*GPU Speedup; MIC Speedup*). The goal is to use CPU cores to execute tasks that have low performance in all other devices. Finally, when a tasks is requested, the head of the queue relative to that device (highest speedup) is retrieved, and the task selected is removed from other queues. To quickly access and remove the task from other queues, we maintain a pointer to its location in the neighbor queue (see Figure 5).

The performance-aware scheduling policies we proposed use speedup estimates to maintain the order of tasks in the queues. Thus, if speedup estimates are accurate enough to not modify the order of tasks in the queues, the performance of scheduler will not be affected. As we present in the experimental evaluation, the time-based scheduling policies (e.g., Heterogeneous Earliest Finish Time (HEFT) tend to be more sensitive to these estimates, requiring more accurate estimates. It is also important to notice that if all tasks ready for execution are more efficiently executed by a single device, other processors will still be used to compute those tasks whenever they become idle. However, our policies will select the task that benefits the most from the available device.

V. Experimental Evaluation

The experimental evaluation was carried out using a distributed memory cluster machine called Stampede. Each node on this Linux cluster is equipped with a dual socket Intel Xeon E5-2680 CPU, 32GB of main memory, and at least one an Intel Xeon Phi SE10P accelerator. It also includes 128 nodes with the an Intel Phi and a NVIDIA K20 GPU, which are primarily used for visualization. The nodes are connected using Mellanox FDR Infiniband network. The codes used in our evaluation were compiled using Intel Compiler 13.1 and “-O3” optimization flag, and Intel Phi based codes run on offload mode. The set of images used as input are those collected in our brain tumor studies [10], which are sliced into 4K×4K tiles for parallel computation.

For sake of evaluation, we have also developed other scheduling policies to compare with our proposed performance-aware scheduling: First Come, First Served (FCFS) and Heterogeneous Earliest Finish Time (HEFT). FCFS keeps a single queue of tasks, which are dispatched for execution in a demand-driven basis in the order they are created. HEFT uses a queue for each processing unit employed (CPU core, GPU, or MIC). The task distribution across queues is computed according to the processing capacity of each unit, based on expected execution time of previous tasks. It maintains a history of execution times and, thereby, assigns a task to the processing unit that minimizes the finish time of that task. HEFT is also implemented in other systems, such as StarPU [14].

A. Evaluating Schedulers Performance

Our first set of experiments evaluates the cooperative execution on a CPU-GPU-MIC environment with regards to the scheduler used. The application is executed using 1 GPU, 1 MIC, and 14 CPU cores for computation: the remaining 2 CPU cores are used to manage the accelerators. The application workload consists of about 800 image tiles, which generates 10,800 fine-grained tasks for execution. Additionally, we varied the number of coarse-grained pipeline stages concurrently active with a Worker (Worker Request Window Size) to evaluate the impact of the decision space to the performance of the schedulers. The window size is increased from 16 (number of processors used for computation: CPU cores + accelerators) until it has negligible impact to the performance. Speedups and execution times used as input to the schedulers were collected in previous runs on a subset of the data.

The performances of the application for FCFS, HEFT, PADAS, and PAMS are presented in Figure 6. As shown, FCFS and PADAS attained similar execution times, with a slightly better performance of the latter. Additionally, the variation of the window size had little impact on FCFS and PADAS. Further, HEFT has achieved poor performance with small window sizes, but was able to outperform FCFS and PADAS with window sizes higher than 45. A large decision space (window size) is important for HEFT, because it allows for the scheduler to better balance work among processors within a node and to reduce miss-assignment of tasks.

Finally, the PAMS scheduler proposed in this work reached shorter execution times than other polices for all window sizes. For instance, in the configuration with window of 80, PAMS is about 1.16× faster than HEFT. It is also noticeable that PAMS is less sensitive to window sizes. It is desirable for a scheduling policy to attain high performance with small window sizes, since this metric directly affects the amount of load imbalance observed among Workers assigned for execution with different nodes of a distributed memory machine. In the next section, we present a detailed study of the scheduling policies.

B. Understanding the Behavior of Schedulers

In order to better understand the performance of each scheduler, we have profiled the task assignment decisions of each policy to correlate it with the performance of tasks on each device. The relative performance and weight of operations to the overall application execution time are presented in Figure 2. It is important to recall that: (i) a high performance variability exists among operations, because of their different computation intensity and data access patterns, making operations more suitable for different processors, and (ii) operations have a different weight to the execution and, as such, miss-assignment of operations will have different impact to the performance.

The profile of the tasks/operations assigned for each of the available processing units (CPU, GPU and MIC) according to the scheduling policy used: FCFS, PADAS, HEFT, and PAMS, for a fixed window size of 80 is presented in Figure 7. First, it is shown that the FCFS assignment to computing devices is nearly the same for all operations. As such, FCFS is not able to take advantage of performance variability among operations. The PADAS scheduler computed a scheduling in which devices were used for different tasks, but it lead to little performance improvement as compared FCFS. The analysis of scheduling of operations with higher execution times (PreWatershed, RBC and ReconNuclei) shows that PADAS correctly used the GPU for PreWatershed, but a significant amount of PreWatershed tasks were also computed using CPU that is inefficient for this operation. Further, PADAS computed most of RBC tasks using CPU, whereas it is more appropriate for execution with the accelerators. The evaluation of the Sobel operation also shows that PADAS was not able to perform a good distribution of tasks, since it mostly used GPU but ReconNuclei is better suited for MIC.

The task assignment profile of HEFT is presented in Figure 7(c). First, we may notice that HEFT has executed most of tasks with short execution times using CPU cores, independently of their performance/speedup on available devices. In addition, HEFT has assigned the PreWatershed tasks exclusively to accelerators, but GPU is twice faster than MIC for this operation and, as a consequence, PreWatershed should be executed with higher percentages using GPU. Since this task represents about 40% of the application execution time, its correct assignment is of major importance to attain good performance. Finally, the profile of PAMS shows that it (i) has correctly assigned most of PreWatershed to GPU; (ii) used MIC to compute tasks for which this device is efficient, such as ColorDevonv, FillHoles, Gradient, PixelIntensity, and Sobel; and, (iii) CPU was used in operations not as efficiently executed by accelerators, including AreaThreshold and BWLabel.

C. Varying the Processors Configuration

This section evaluates the schedulers with different processors configurations: CPU-GPU (15 CPU cores and 1 GPU), CPU-MIC (15 CPU cores and 1 MIC) and CPU-GPU-MIC (14 CPU cores, 1 GPU, and 1 MIC). The workload used is the same employed in previous sections.

The execution times of the schedulers are presented in Figure 8. As shown, FCFS attains the worst performance among for all configurations. In cooperative executions using two types of processors, CPU-MIC or CPU-GPU, PADAS and PAMS achieved similar execution times. In this configuration, both schedulers basically perform the same scheduling using different data structures: a single queue ordered by speedup vs. two queue sorted by speedups. Additionally, PADAS and PAMS have better performance than HEFT in the same settings. Further, in the configuration with CPU-GPU-MIC, PAMS outperform all other scheduling strategies as discussed in previous sections. It is also important to highlight that the use of GPU always lead to good performance, e.g., best CPU-GPU execution time is about 1.27× faster than best CPU-MIC execution time.

D. Sensibility to Inaccurate Estimations

This section evaluates schedulers with regards to impact of accuracy in input metrics (execution time and speedup) to schedulers performance. For sake of this evaluation, we have conducted an experiment in which errors were inserted in a controlled manner to expected execution times of operations. For schedulers using speedup, this value was derived from execution times with errors. Additionally, we varied the error inserted in 0%, 10%, 25%, 50%, 75%, and 100% of the expected execution times, with equal chance of the error inserted be an increasing or a decreasing of the execution time.

The performance of the schedulers is presented in Figure 9. As shown, the performance of speedup based schedulers, PADAS and PAMS, are little impacted as error rates are increased. In contrast, the HEFT time based scheduler dramatically suffers with error rate increases. HEFT is even slower than FCFS for error rates higher than 50%. These results are important to show that the performance scheduling strategies based on speedup are very insensitive to error estimates, and still working well even with very high error rates. The PAMS with 0% error rate is only 1.07× faster than PAMS with 100% error rate.

E. Scalability Results

This section evaluates the application using a distributed memory cluster machine. The example application, described in Section II, is a hierarchical pipeline with first level having segmentation and feature computation stages, with each of the stages being implemented as another pipeline of fine-grain operations. We evaluate the CPU-only version of the application, which uses all 16 CPU cores in each node, and cooperative versions using CPUs, GPUs, and MICs with differ scheduling strategies.

The strong scaling evaluation is presented in Figure 10. The experiments used 6,379 4K×4K image tiles as input. All versions of the application scaled well, and the best cooperative executions using PAMS are about 2.2× faster than the CPU-only executions for all number of nodes. Additionally, as presented, PAMS is nearly 1.2× faster than HEFT, and the improvements are maintained as the number of nodes is increased.

VI. Related Work

The efficient execution on hybrid systems equipped with CPUs and accelerators is a challenging problem, which requires, for instance, optimized implementation of application operations for multiple processors and scheduling of work among heterogeneous devices. Recently, a number of runtime systems [2], [3], [4], [5], [6], [15], [16], [8], [17], [18], compiler techniques [17], and domain-specific libraries [19] have been proposed to reduce the programming effort and complexity involved in porting applications to these systems.

Execution on distributed CPU-GPU platforms has been the target of a number of projects [5], [6], [15], [16], [8], [17], [18], [20]. Ravi et al. [17] developed compiler based translation of generalized reductions to CPU-GPU systems. DaGuE [6] and StarPU [5] express computations as a Directed Acyclic Graph (DAG) and were evaluated for linear algebra applications. OmpSs [8] also executes data flow applications, which are parallelized via compiler from user annotated code. XKaapi [7] also supports cooperative execution on CPU-GPU machines using a multiversioning scheme in which operations may have multiple implementations targeting different computing devices.

The study of applications performance on the Intel's Xeon Phi co-processor is another increasingly important topic of research [21], [22], [23], [24]. Linear algebra algorithms have been implemented for the MIC [22], [24], as well as the Lattice Quantum

Chromodynamics (LQCD) method [21]. In [25], the authors evaluate the sharing of coprocessors among multiple machines by adding support for offloading computation to an Intel Phi on a remote node. A detailed evaluation of the overheads and benefits of using OpenMP to parallelize operation to Intel Phi was also recently carried out [23].

In our work, we target the cooperative execution of a scientific data analysis pipeline on distributed hybrid systems equipped with CPUs, GPUs, and MICs. The motivating application is developed as a hierarchical pipeline, which allows for fine-grain operations in coarse-grain stages to be exported for scheduling on devices in a node. Most of the previous works deal with task mapping for applications in which operations attain similar speedups when executed on a GPU vs a CPU. In contrast to previous research, we argue that performance variability of internal tasks of an application is crucial to attain high performance. Our proposed performance-aware scheduling policies have shown to be significantly more efficient than other competitive scheduling techniques, such as HEFT, for hybrid systems equipped with CPUs and accelerators. Additionally, we expect that our evaluation on cooperative execution using CPUs and multiple accelerators may provide clues on the potential benefits of cooperative execution on state-of-the-art accelerators. To the best of our knowledge, this is the first work to investigate the cooperative use of CPUs, GPUs, and MICs.

VII. Conclusions and Future Directions

In this work, we evaluated the execution of a pathology image analysis application on hybrid systems equipped with CPUs, GPUs, and MICs. We proposed two performance-aware scheduling techniques, which take into account performance variabilities across different tasks to better utilize hybrid systems. Our proposed strategies were implemented in the Extreme DataCutter runtime system, and were used to schedule fine-grain tasks created with each stage of the application across heterogeneous devices.

We experimentally evaluated our proposed approach with 800 image tiles, which generates 10,800 fine-grained tasks for execution. We compared our proposed schedulers with two well known scheduling policies: FCFS and HEFT. Our results showed that the PAMS scheduler achieved shorter execution times than the other polices, being 1.16× faster than HEFT, which presents the second shortest execution time. We also showed that our proposed performance-aware strategies, PADAS and PAMS, are less sensitive to inaccuracy in performance estimation data, in contrast to the HEFT time based scheduler, which suffers dramatically as the error increases. Finally, our experiments showed that the application scales well with EDC and the best cooperative executions, using PAMS, are about 2.2× faster than the CPU-only executions for any number of nodes.

As future work, we intend to improve our proposed performance-aware PAMS and PADAS taking advantage of the insights gained from the contributions of this work, to further enhance the efficiency of pathology image analysis application. And also evaluate PAMS and PADAS in others data flow applications.

Acknowledgments

This research was partially supported by the NSF through XSEDE resources provided by the XSEDE Science Gateways program, part by HHSN261200800001E from the NCI, by R01LM011119-01 and R01LM009239 from the NLM, and part by CNPq, CAPES, FINEP, Fapemig, and INWEB.

References

1. He B, Fang W, Luo Q, Govindaraju NK, Wang T. Mars: A MapReduce Framework on Graphics Processors. Parallel Architectures and Compilation Techniques. 2008
2. Linderman MD, Collins JD, Wang H, Meng TH. Merge: a programming model for heterogeneous multi-core systems. SIGPLAN Not. 2008; 43(3):287–296.
3. Luk, C-K.; Hong, S.; Kim, H. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. 42nd Int. Symp. on Microarchitecture; 2009.
4. Rossbach, CJ.; Currey, J.; Silberstein, M.; Ray, B.; Witchel, E. PTask: operating system abstractions to manage GPUs as compute devices. The 23rd ACM Symposium on Operating Systems Principles; 2011. p. 233-248.
5. Augonnet, C.; Aumage, O.; Furmento, N.; Namyst, R.; Thibault, S. StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. The 19th European MPI Users' Group Meeting (EuroMPI 2012); 2012.
6. Bosilca G, Bouteiller A, Herault T, Lemarinier P, Saengpatsa N, Tomov S, Dongarra J. Performance Portability of a GPU Enabled Factorization with the DAGuE Framework. IEEE Int Conf on Cluster Comp. 2011:395–402.
7. Gautier, T.; Lima, JVF.; Maillard, N.; Raffin, B. Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. IPDPS '13: Proceedings of the 2013 IEEE International Symposium on Parallel and Distributed Processing; IEEE Computer Society; 2013. p. 1299-1308.
8. Bueno, J.; Planas, J.; Duran, A.; Badia, R.; Martorell, X.; Ayguade, E.; Labarta, J. Productive Programming of GPU Clusters with OmpSs. 2012 IEEE 26th Int. Parallel Distributed Processing Symp. (IPDPS); 2012. p. 557-568.
9. Teodoro, G.; Pan, T.; Kurc, T.; Kong, J.; Cooper, L.; Podhorszki, N.; Klasky, S.; Saltz, J. High-throughput analysis of large microscopy image datasets on cpu-gpu cluster platforms. 27th IEEE Int. Parallel and Distributed Processing Symp.; 2013. p. 103-114.
10. Cooper L, Kong J, Gutman D, Wang F, et al. An integrative approach for in silico glioma research. IEEE Trans Biomed Eng. 2010; 57(10):2617–2621. [PubMed: 20656651]
11. Teodoro, G.; Kurc, T.; Kong, J.; Cooper, L.; Saltz, J. Comparative Performance Analysis of Intel Xeon Phi, GPU, and CPU: A Case Study from Microscopy Image Analysis. 28th IEEE Int. Parallel and Distributed Processing Symp.; 2014.
12. Teodoro G, Pan T, Kurc T, Kong J, Cooper L, Saltz J. Efficient Irregular Wavefront Propagation Algorithms on Hybrid CPU-GPU Machines. Parallel Computing. 2013
13. Beynon MD, Kurc T, Catalyurek U, Chang C, Sussman A, Saltz J. Distributed processing of very large datasets with DataCutter. Parallel Comput. 2001; 27:1457–1478.
14. Augonnet C, Thibault S, Namyst R, Wacrenier P-A. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Euro-Par '09. 2009
15. Teodoro, G.; Hartley, T.; Catalyurek, U.; Ferreira, R. Runtime optimizations for replicated data flows on heterogeneous environments. The 19th ACM International Symposium on High Performance Distributed Computing; 2010. p. 13-24.
16. Teodoro G, Sachetto R, Sertel O, Gurcan M, WM, Catalyurek U, Ferreira R. Coordinating the use of GPU and CPU for improving performance of compute intensive applications. IEEE Cluster. 2009:1–10.
17. Ravi V, Ma W, Chiu D, Agrawal G. Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. The 24th ACM Int Conf on Supercomputing. 2010:137–146.

18. Lima, JVF.; Broquedis, F.; Gautier, T.; Raffin, B. Preliminary experiments with xkaapi on intel xeon phi coprocessor. 25th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD; 2013. p. 105-112.

19. Bradski G. The OpenCV Library. Dr Dobb's Journal of Software Tools. 2000

20. Teodoro G, Hartley T, Catalyurek U, Ferreira R. Optimizing data flow applications on heterogeneous environments. Cluster Computing. 2012; 15:125–144.

21. Joó B, Kalamkar D, Vaidyanathan K, Smelyanskiy M, Pamnany K, Lee V, Dubey P, Watson I, William. Lattice QCD on Intel Xeon Phi Coprocessors. Supercomputing, ser LNCS. 2013; 7905:40–54.

22. Heinecke, A.; Vaidyanathan, K.; Smelyanskiy, M., et al. Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel Xeon Phi Coprocessor. The 27th IEEE International Symposium on Parallel Distributed Processing; 2013.

23. Cramer, T.; Schmidl, D.; Klemm, M.; an Mey, D. OpenMP Programming on Intel Xeon Phi Coprocessors: An Early Performance Comparison. Many-core Applications Research Community Symposium; Nov 2012;

24. Eisenlohr J, Hudak DE, Tomko K, Prince TC. Dense Linear Algebra Factorization in OpenMP and Cilk Plus on Intel's MIC Architecture: Development Experiences and Performance Analysis. TACC-Intel Highly Parallel Computing Symp. 2012

25. Hamidouche, K.; Potluri, S.; Subramoni, H.; Kandalla, K.; Panda, DK. MIC-RO: enabling efficient remote offload on heterogeneous many integrated core (MIC) clusters with Infini-Band. 27th Int. ACM International Conference on Supercomputing, ser. ICS '13; 2013. p. 399-408.

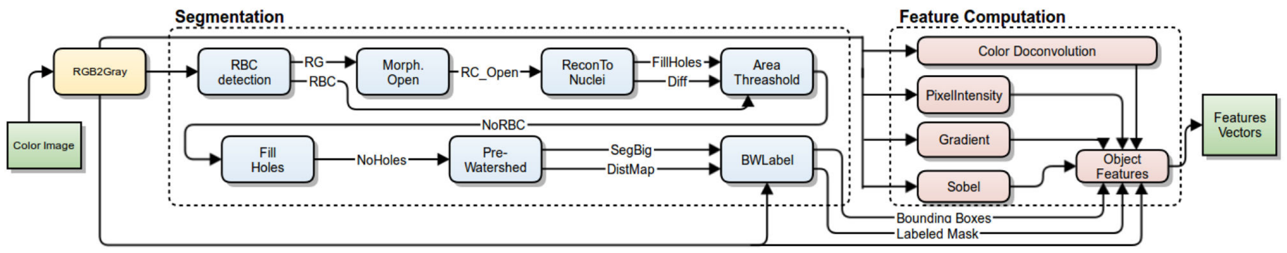

Fig. 1.

Pathology image analysis application data flow. The Segmentation and Feature Computation (coarse-grain) stages of the application are decomposed into another graph of finer-grain operations, which are assigned for execution with heterogeneous devices.

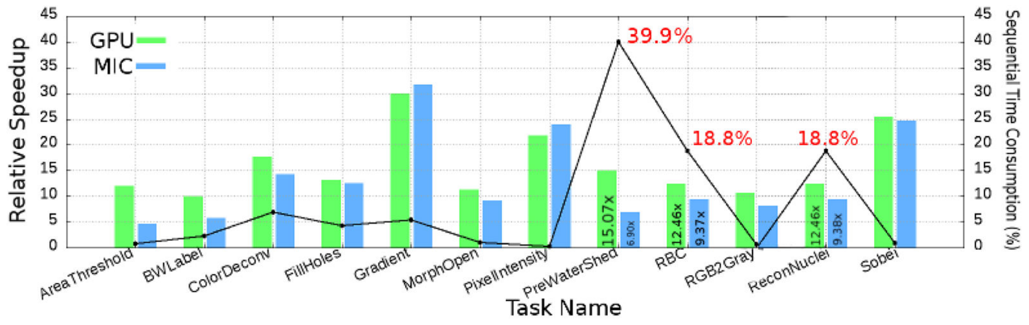

Fig. 2.

Speedups of the application fine-grain operations when executed with accelerators, using the single CPU core as a reference. The percentage of the operations execution time (weight) with regards to the whole application execution time is also presented.

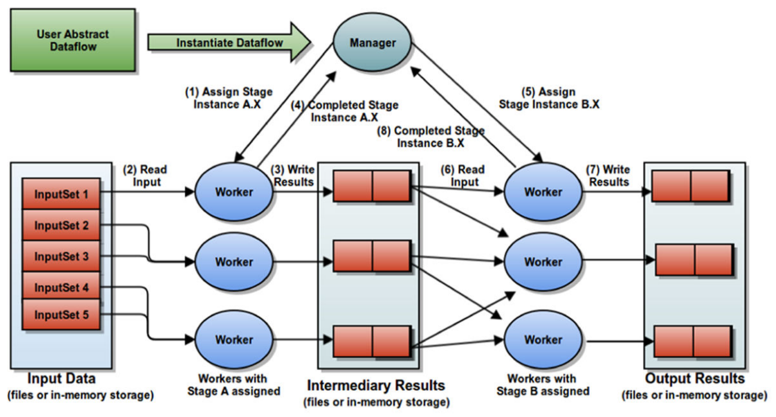
User Abstract Dataflow
Instantiate Dataflow
Manager
(1) Assign Stage Instance A.X
(4) Completed Stage Instance A.X
(8) Completed Stage Instance B.X
(5) Assign Stage Instance B.X
(2) Read Input
(3) Write Results
(6) Read Input
(7) Write Results
InputSet 1
InputSet 2
InputSet 3
InputSet 4
InputSet 5
Worker
Worker
Worker
Worker
Worker
Worker
Input Data
(files or in-memory storage)
Workers with Stage A assigned
Intermediary Results
(files or in-memory storage)
Workers with Stage B assigned
Output Results
(files or in-memory storage)

Fig. 3.

Extreme DataCutter Architecture Overview.

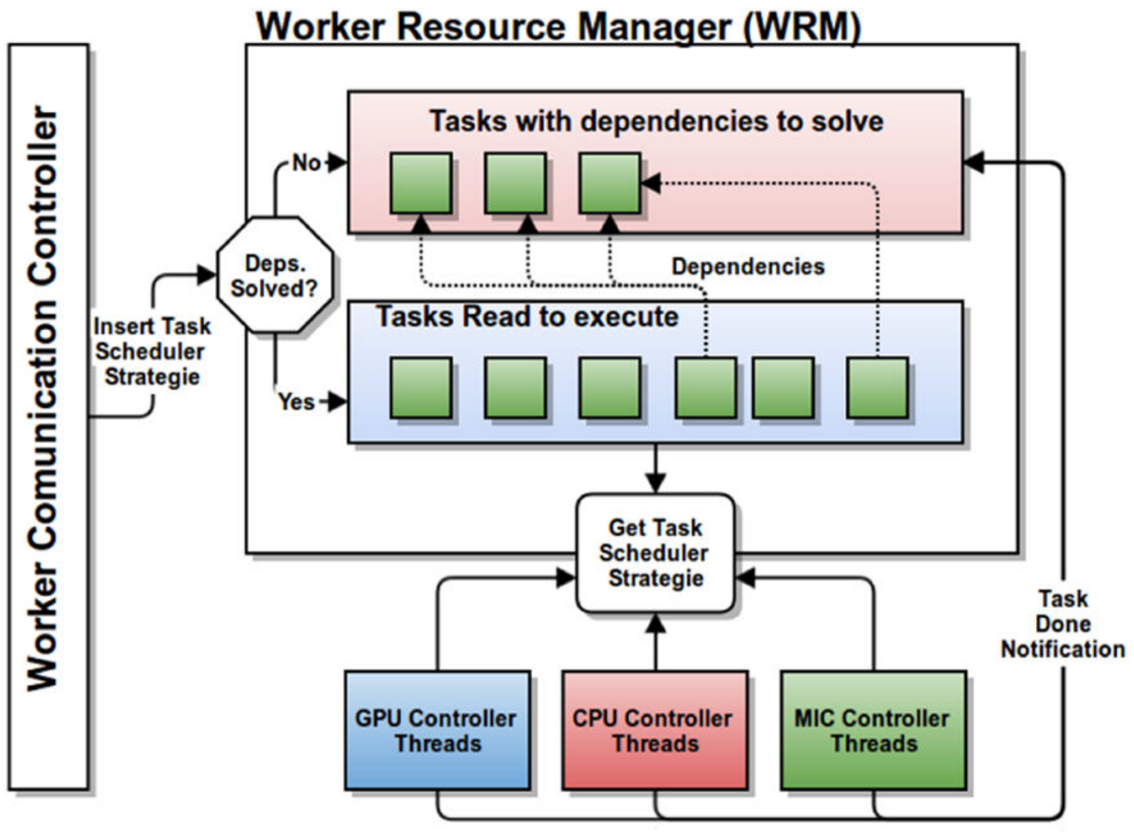
Worker Resource Manager (WRM)
Worker Comunication Controller
Tasks with dependencies to solve
No
Deps. Solved?
Yes
Insert Task Scheduler Strategie
Dependencies
Tasks Read to execute
Get Task Scheduler Strategie
Task Done Notification
GPU Controller Threads
CPU Controller Threads
MIC Controller Threads

Fig. 4.
Worker Process Environment.

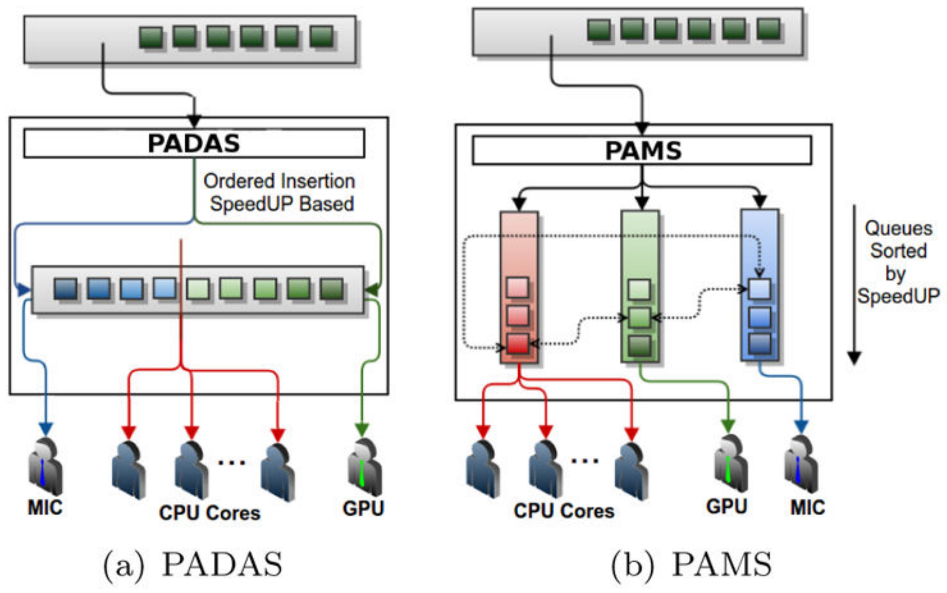

Fig. 5.
Proposed Performance-Aware Scheduling Strategies

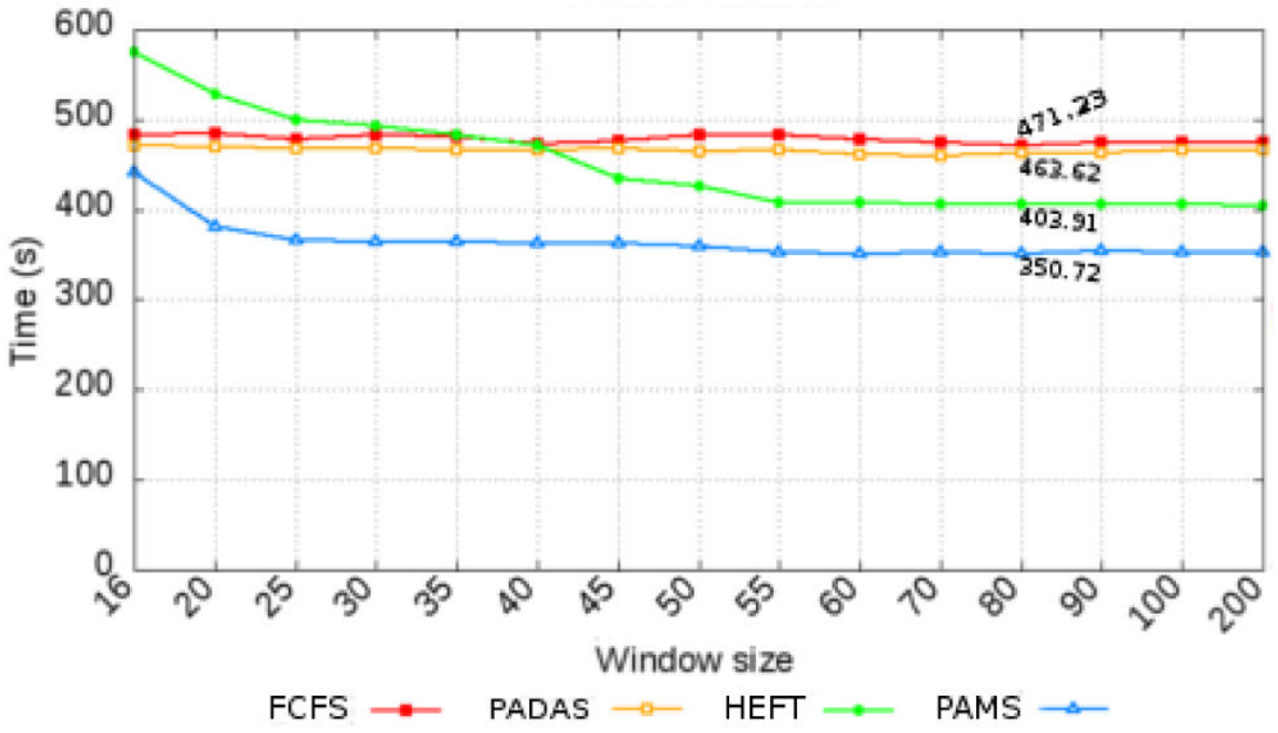

Fig. 6.
Performance of schedulers as the number of pipeline tasks concurrently assigned to a Worker (Worker Request Window Size) varies.

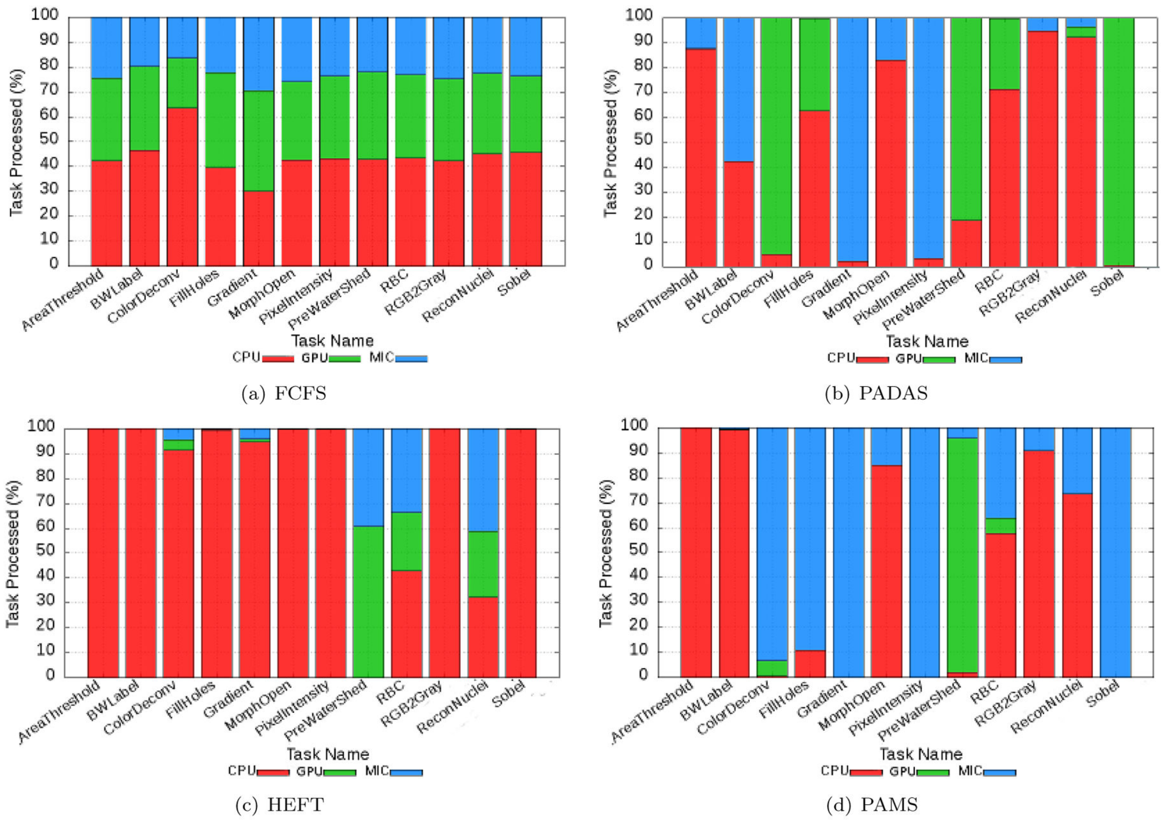

Fig. 7.

Profile of task assignment to devices for each of the tasks/operations executed in our application, fixed window size of 80, and different scheduling policies.

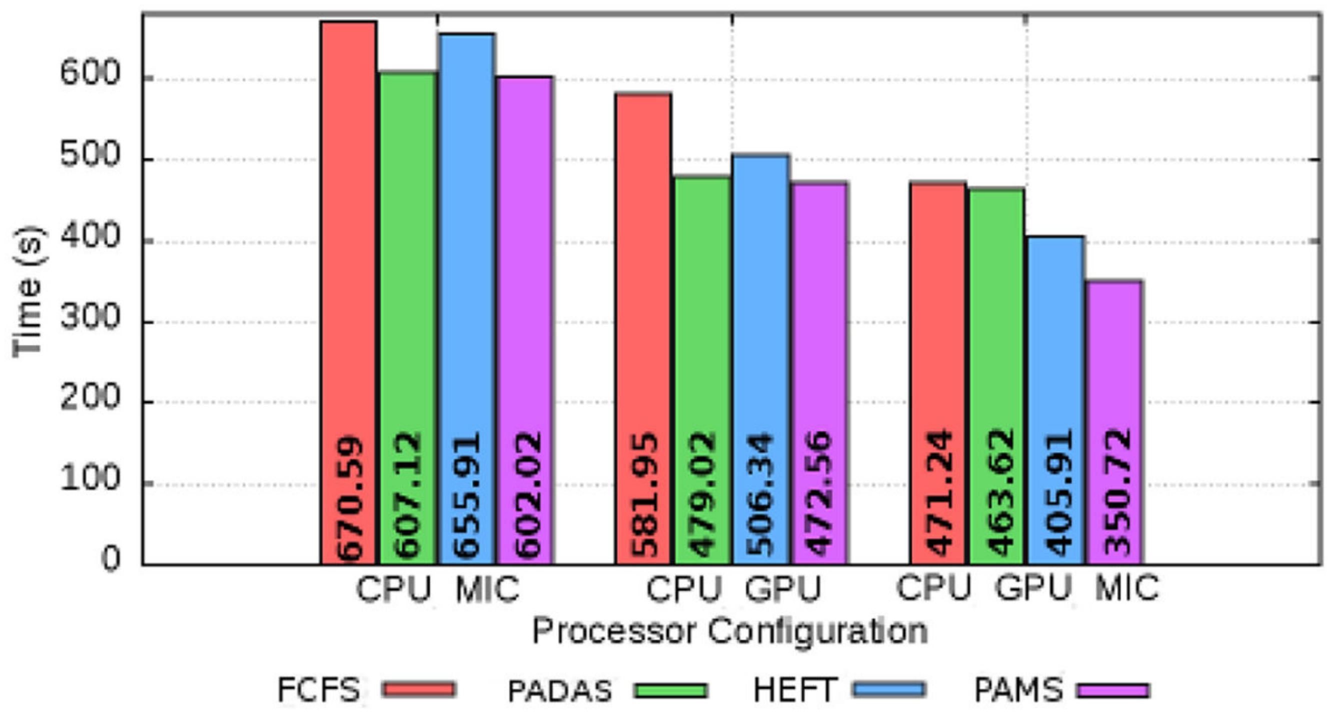

Fig. 8.

Performance of schedulers in cooperative executions using different types of processors.

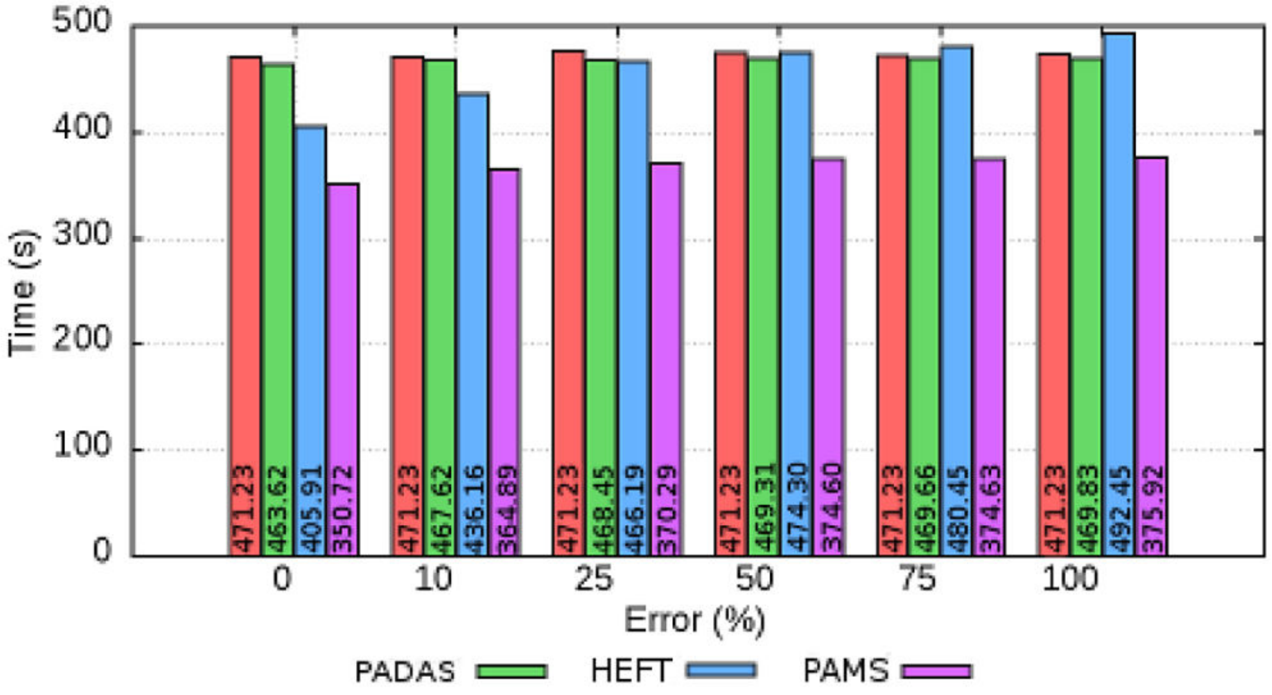

Fig. 9.

Performance of scheduling policies as different error levels are inserted in estimated execution times of operations.

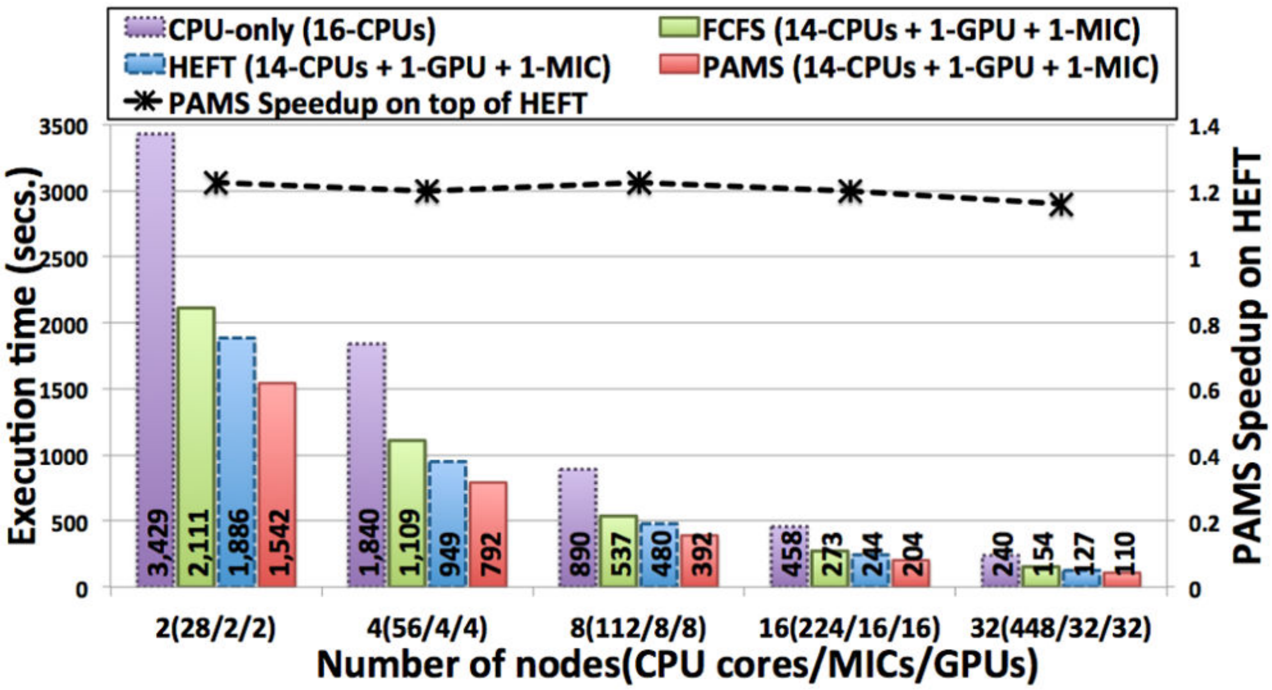

Fig. 10.
Multi-node strong-scaling evaluation.