

Coordinating the Use of GPU and CPU for Improving Performance of Compute Intensive Applications

George Teodoro¹, Rafael Sachetto¹, Olcay Sertel²,
Metin N. Gurcan², Wagner Meira Jr.¹, Umit Catalyurek², Renato Ferreira¹

¹Department of Computer Science
Universidade Federal de Minas Gerais, Brazil
{george, sachetto, meira, renato}@dcc.ufmg.br

²Department of Biomedical Informatics
The Ohio State University, USA
{osertel, gurcan, umit}@bmi.osu.edu

Abstract—GPUs have recently evolved into very fast parallel co-processors capable of executing general purpose computations extremely efficiently. At the same time, multi-core CPUs evolution continued and today’s CPUs have 4-8 cores. These two trends, however, have followed independent paths in the sense that we are aware of very few works that consider both devices cooperating to solve general computations.

In this paper we investigate the coordinated use of CPU and GPU to improve efficiency of applications even further than using either device independently. We use Anthill runtime environment, a data-flow oriented framework in which applications are decomposed into a set of event-driven filters, where for each event, the runtime system can use either GPU or CPU for its processing. For evaluation, we use a histopathology application that uses image analysis techniques to classify tumor images for neuroblastoma prognosis. Our experimental environment includes dual and octa-core machines, augmented with GPUs and we evaluate our approach’s performance for standalone and distributed executions.

Our experiments show that a pure GPU optimization of the application achieved a factor of 15 to 49 times improvement over the single core CPU version, depending on the versions of the CPUs and GPUs. We also show that the execution can be further reduced by a factor of about 2 by using our runtime system that effectively choreographs the execution to run cooperatively both on GPU and on a single core of CPU. We improve on that by adding more cores, all of which were previously neglected or used ineffectively. In addition, the evaluation on a distributed environment has shown near linear scalability to multiple hosts.

This work was partially supported by INCTWeb, by the Children’s Neuroblastoma Cancer Foundation Young Investigator Award to Metin Gurcan, and by DOE Grant DE-FC02-06ER2775 and by NSF Grants CNS-0643969.

I. INTRODUCTION

Histopathological examination is an important step for diagnosis and evaluating prognosis of patients with cancer diagnosis. However visual examination performed by pathologists under a microscope is tedious and subject to considerable inter- and intra-reader variations [1]. Using image analysis, it is possible to extract quantitative measurements that provides more precise description of tumor morphology; therefore it may lead to more accurate and consistent predictions of clinical outcomes.

We are developing computer-aided prognosis systems for the automated analysis of digitized neuroblastoma whole-slide tissue samples [2]. Neuroblastoma (NB) is a cancer of the nervous system mainly affecting infants and children. The current gold standard of NB prognosis is the visual evaluation of biopsy samples to identify certain morphological characteristics such as the degree of Schwannian stromal development, the grade of differentiation, and the mitosis and karyorrhexis index, with microscopic examinations of tumor tissue [3].

The resulting digitized whole-slide images are quite large with their sizes up to 40GB, which requires efficient computational tools to handle the image analysis pipeline. As we discuss later, the idea is to analyze coarse-grain images and, whenever necessary, perform finer-grain analyses of portions of the image. The cost of the analysis of a single image, coupled with this progressive refinement type of iterations make the application very computational intensive.

The computational cost of this Neuroblastoma Image Analysis (NBIA) application motivates its

parallelization, but such effort has to deal with some challenges, such as the large volume of data, which may not fit in the available machines, and the irregularity of the application, since the number of refining iterations over an image, and thus the associated cost, is data dependent.

We have been witnessing a growth of heterogeneous platforms (CPUs and GPUs) in terms of both computational power and applications being parallelized for those platforms. Coordinated and efficient use of both types of processing units is still a challenge, mainly for irregular applications such as NBIA. The challenge in this case lies on how to gather the best from each processing unit, since CPU cores support fewer threads and do not require additional data communication, while GPUs expect large number of concurrent threads, require explicit communication between the standard machine memory and the GPU, and are better suited for SIMD computations. Despite the difficulty of such scheduling decision, NBIA and other similar applications require that the decision is taken instantaneously, considering the current load on each processing unit and the characteristics of the tasks to be processed at the scheduling moment.

In this paper we focus on a scalable and efficient parallelization of a multi-resolution system [2], which is not only a relevant and realistic application, but also an interesting case study in terms of how to better exploit heterogeneous architectures. These are the contributions presented in this paper:

- Event-oriented parallelization of the NBIA application for the Anthill programming model, detailed in Section II, and its instantiation on a heterogeneous platform comprising both CPUs and GPUs.
- Design and evaluation of task scheduling strategies, which allow an adaptive and proactive exploitation of computational resources available in heterogeneous architectures.
- Performance improvements in the order of 100%, achieved using both CPUs and GPUs as well as the scheduling strategies proposed, on top of pure GPU versions.

The remainder of the paper is organized as follows. In the next section we briefly describe the relevant details of Anthill and the modifications we built to allow multiple event handlers for different devices. In Section III, we describe more details on NBIA application and our port to Anthill. Section IV describe our extensive experimental evaluation of the application on different alternative execution scenarios and scheduling policies.

We discuss some relevant related bibliography in Section V and present our conclusions and future directions in Section VI.

II. ANTHILL

Anthill [4], [5] is a runtime framework inspired by the filter-stream programming model implemented in Datacutter [6]. This model uses a data-flow approach to decompose the applications into a set of processing units referred to as filters. Data is transferred through the filters using streams which allow fixed-size untyped data buffers to be transferred from one filter to the next.

The application decomposition process leads to task parallelism at runtime, as the application becomes a set of filters connected like a directed graph. At execution time, multiple copies of each filter that compose the application are instantiated on several machines of the system and the streams are connected from sources to destinations. The transparent copy mechanism allows any vertex of the application's graph to be replicated over many compute nodes and the data that goes through each filter may be partitioned among the copies creating data parallelism.

For several applications, the natural decomposition is a cyclic graph, where the execution consists of multiple iterations over the filters. An application starts with data representing an initial set of possible solutions and as these pass through the filters, new candidate solutions are created. In our experience with developing applications in Anthill we noticed that this behavior also leads to asynchronous executions, in the sense that several candidate solutions (possibly from different iterations) are processed simultaneously at run-time.

Anthill, therefore, tries to exploit the maximum parallelism in applications by using all three possibilities discussed above: task parallelism, data parallelism and asynchrony. By dividing the computation into multiple pipeline stages, each one replicated multiple times, we can have a very fine-grained parallelism and, since all this is happening asynchronously, the execution is mostly bottleneck free.

With regard to the transparent copy mechanism, care should be taken when the filter being replicated has state. Ideally, that state should be partitioned. For that to happen, the incoming messages to any filter need to be sorted with respect to the state variables they update and delivered to the appropriate transparent copies. The mechanism that guarantees it in Anthill is called *Labeled Stream*. A label is associated with each message as they

are sent down the stream, which checks that label against a bijective (e.g., hash) function that selects one copy to deliver the message to. This guarantees that messages associated with the same label, and consequently the same state variables, are delivered always to the same transparent copy.

A. Filter Programming Abstraction

A filter processes data as it flows through, possibly changing the nature of the data that is passed to the next filter. On those pretenses, a programmer, when defining a filter, should have to worry only with the transformation function, which is executed for each data unit in turn, and that may internally cause some data to be forwarded to the next filter. It is up to the filter run-time environment to decide, once an event is raised, when it may be processed, given the hardware resources available. As long as there are resources available and events pending, execution may continue.

A filter may be built so that it receives input from multiple, independent, streams. The programmer indicate what should be done upon arrival of data on any stream, even with different actions for each source. The filter run-time environment controls any non-blocking I/O issues that are necessary, and chooses the right processing function for each event as defined by the programmer. Such approach derives heavily from the message-oriented programming model pioneered by the *x*-kernel [7] and later extended to include explicit, user-defined events in Coyote [8].

Anthill included several modules to support event-driven programming. As depicted in Figure 1, there is an event layer that is divided into three components: Communication module, Event queue controller, and Event handler. We discuss each of them next.

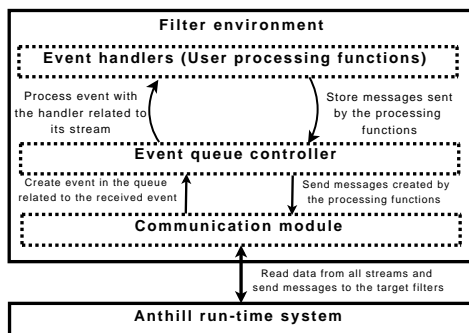


Fig. 1. An Overview of Anthill Event-Driven Runtime System

- **Communication module:** responsible for monitoring all input streams and sending mes-

sages for the target filter instances. Its operation essentially consists of creating an event in the respective event queue, whenever a message associated with a stream arrives and the dependencies are satisfied.

- **Event queue controller:** responsible for controlling the event queues associated with each filter instance. This component determines in which order and when the events in each queue should be processed and calls the appropriate filter processing function.
- **Event handler:** it contains the application-specific code. For each input stream, the developer creates a function that processes the data received through it and registers it in the system. Whenever the event associated with a given stream is triggered, it is that registered function which gets executed, receiving as input the event to be handled (*i.e.*, the data to be processed).

By keeping those elements apart, the application programmer is isolated from low-level abstractions and control is assigned to the run-time system. That makes it easier for scheduling and load balancing to be implemented without affecting the programmer's code. Special scheduling can be created by monitoring the communications module and the event queue controller, and by actuating back on those modules, without requiring changes to the application code.

B. Supporting Heterogeneous Resources

In this section we describe our approach towards transparently exploiting heterogeneous resources available in a distributed memory machine. As mentioned earlier, the idea is to allow the programmer to provide different handlers for the same stream, each to be executed on a different component of the compute node. We integrate additional modules to the filter run-time framework which allow Anthill to automatically choose among the available alternatives and dispatch the execution of events to different components concurrently.

The event driven model fits nicely into this scenario for at least three reasons: (1) the input/output are well defined, so the handler functions can be implemented in a high-level language appropriate for the device it is targeting, (2) the run-time system can use any device to process events from the same stream, and (3) the run-time system can choose on the fly, when the input dependencies are ready, the appropriate device to process an event.

The decision space for the scheduler within this extended Anthill is broad and involves more than

just decisions about when and how many events should be dispatched in parallel. First of all, since a heterogeneous set of processors may be available, the system must consider for which devices an event handler has been defined, once the user does not have to provide handler implementations for all devices. Then, it must consider performance trade-offs between the different implementations available, which may even be data-dependent. Finally, it must consider the overall coordination of the entire execution for sake of flow optimization.

Given such considerations, the decisions are taken on a per event basis, and may change for different events, even if they are of the same kind. This allows filters which have handler implementations to different devices to be scheduled according to their availability, creating a very flexible environment. Considering a distributed memory machine with GPUs, Anthill is capable of scheduling events to GPUs on any node, for any filter that has a GPU-targeted implementation. If a filter has both implementations, Anthill can schedule events concurrently to both CPU and GPU transparently.

In our current prototype implementation, there are two event scheduling policies: (i) first-come first-served (FCFS), and (ii) dynamic weighted round robin (DWRR). The FCFS is the default one and the next event to be processed will be the oldest queued, and it will be dispatched to the first device available for which there is an implementation of the event handler. The second policy, DWRR, divide the events to be processed according to a weight that may vary during the execution time, and receive a value for each device. During the execution, this weight is then used to order the input queue of each device, and, when a certain processor is available, the event to be processed is the highest weighted for it.

At this moment, we target CPU and GPU codes only, while the data transfers and dependencies are handled transparently by the run-time system. However, our run-time framework can easily be extended to support other devices available in a compute node, such as an FPGA or Cell co-processor.

To implement these additional functionalities, we extended Anthill’s API to accept the association of event handlers to different hardware devices and some wrapper functions to isolate the programmer from the device peculiarities (e.g., memory management). Figure 2 depicts the extended architecture. We basically create two modules on top of the filter run-time framework: the device scheduler and the event executor.

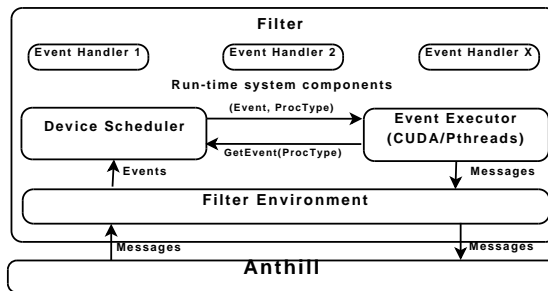


Fig. 2. Framework Extended Architecture

The device scheduler keeps track of, for each event handler, how many device specific codes are available and, given an event queue, determine which is the next event and where it should be processed. The event executor is a layer of system software that isolates the user provided code from the specific hardware issues. At run-time, the event executor creates threads and associates them with different devices. Each thread contacts the device scheduler, whenever idle, and it determines an event to be processed within that thread.

III. NEUROBLASTOMA

A. Multi-resolution Neuroblastoma Image Analysis System

The goal of neuroblastoma image analysis (NBIA) is to classify the digitized tissue samples into different subtypes that have prognostic significance. In this application, we concentrated on the classification of stromal development as either stroma-rich or stroma-poor, which is one of the morphological criterions in NB prognosis that contributes to the categorization of the histology as favorable and unfavorable [3].

As the slides can have a very high resolution, the whole-slide NBIA first decomposes the image into smaller image tiles and each image tile is processed independently. Due to that, the NBIA employed a multi-resolution image analysis approach [9], which mimics the way pathologists examine the tissue slides under the microscope such that the image analysis starts from the lowest resolution, which corresponds to the lower magnification levels in a microscope and uses the higher resolution representations for the regions where the decision for the classification requires more detailed information.

Without loss of generality, the multi-resolution image analysis operates on an image tile as follows: it first decomposes each image tile from the original image into multiple resolutions representation,

as for example a three layered multi-resolution could be constructed up with (32×32) , (128×128) , and (512×512) . The basic classification strategy starts using the lowest resolution images, and stop at the resolution level where the classification satisfies a pre-determined criterion. The classification is achieved based on statistical features that characterize the texture of tissue structure. Therefore, we first apply a color space conversion and switch to La^*b^* color space, where color and intensity are separated and the difference between two pixel values is perceptually more uniform. The texture information is extracted using co-occurrence statistics and local binary patterns (LBPs), which help characterizing the color and intensity variations in the tissue structure [9]. Finally, the classification confidence at a particular resolution is computed using a hypothesis testing after which either the classification decision is accepted or the analysis resumes with higher resolutions if exists. The result of the image analysis is a classification label assigned to each image tile indicating the underlying tissue subtype, e.g., stroma-rich or stroma-poor, or background. More details on the NBIA can be found in [2].

B. Anthill Implementation

The NBIA implementation in Anthill, as shown in Figure 3, has five filters: (i) Image reader: which is responsible for reading the tiles in the RGB color space and send to the next step; (ii) Color conversion: that receives the RGB image tile and convert it into a La^*b^* color space, (iii) Computation of the statistical features: this filter receives the image, represented in the La^*b^* color space, and computes the feature vector consisting of the co-occurrence statistics and LBPs, (iv) the Classifier filter, which receives the feature vector and applies the classification and the hypothesis testing to decide whether or not the classification is satisfactory, and, finally, (v) the Start/Output filter, that controls the processing flow. It starts sending the tiles to be processed, at lowest resolution, to the Image reader, and as it receives each tile classification from the Classifier filter. When a classification is not satisfactory it switches to the higher-resolution, and send a message to the Image reader with the tileId and new resolution, otherwise it continues with the subsequent image tiles. This loop continues until all tiles classification is satisfactory, or they are computed at highest resolution.

In our approach to exploit heterogeneous resources we implemented the CPU and GPU versions of the Color conversion and Statistical fea-

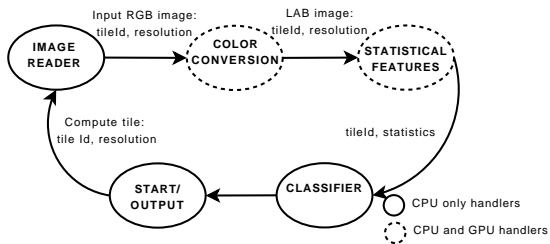


Fig. 3. NBIA filters

tures filters, which are the most computational intensive tasks. Thus, as the streamed data arrives at these filters, the Anthill Device scheduler chooses a specific device, and, consequently, a handler function to process the received data.

C. Application Tasks Analysis and Scheduling

In this section, we analyzed the NBIA relative performance in CPU and GPU as a function of its input granularity. During this evaluation we generated different workloads, using a fixed number of 26,742 tiles, where we varied the input tiles size maintaining a single resolution level. In our evaluation, the experiments were repeated four times, and the maximum standard deviation was 3.3%. The hardware used for these experiments is a dual quad-core AMD Opteron 2.00GHz/NVIDIA GeForce GTX 260, which is described in section IV.

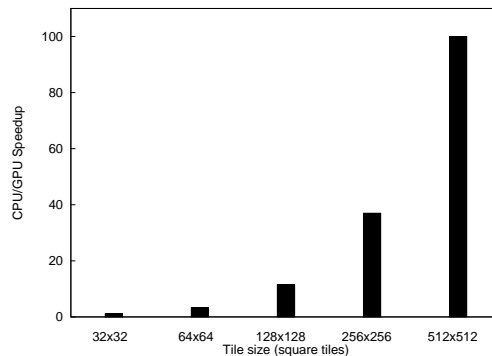


Fig. 4. Application speedup variation

The speedup between the devices, as we increased the tiles resolution, is shown in Figure 4. These results presented a high variation w.r.t. the relative performance between CPU and GPU. While the performance is similar for tiles of 32×32 , the GPU is 100 times faster for 512×512 tiles.

The GPUs poor performance for small tiles occurs due to its under utilization, since small tasks are not demanding enough to fully exploit

its parallel processing capacity, and the use of GPUs implies on communication overhead associated with transferring data to/from the GPU main memory and to invoke the *kernel* function.

In order to improve the devices usage efficiency of a real execution of the NBIA, where multiples processing tasks with different resolution can be concurrently active, we used the relative performance information as an input to our scheduling. Thus when a buffer containing a tile image is received, it is evaluated with a user provided function that receives the event as a parameter and returns the relative speedup. The events are then inserted into the streams queue according to the returned value, and, whenever a device is available, the scheduler can choose the most suitable event to be processed based on its estimated speedup. While the speedup is only used to order the events, for instance, the neuroblastoma simply return the image tile size as the speedup to have the best possible order.

IV. EXPERIMENTAL RESULTS

The experiments with heterogeneous environments were performed using two clusters: (1) 10 PCs connected using a Gigabit Ethernet Switch, where each node has an Intel Core 2 Duo CPU 2.13GHz, 2 GB of main memory, an NVIDIA GeForce 8800GT GPU, and Linux operating system; (2) 4 PCs connected using a Gigabit Ethernet, each node with a dual quad-core AMD Opteron 2.00GHz processor, 16GB of main memory, an NVIDIA GeForce GTX 260, and Linux operating system. Each result shown in this paper is the average of four experiments. During the experiments we used a GPU it has also occupied a CPU core, responsible for managing it.

A. Application performance

In this section, we evaluated the NBIA performance on a single machine of each available hardware configuration. These experiments were performed using input images with a fixed number of 26,742 tiles and two resolution levels: (32x32) and (512x512), as we varied the recalculation rate (the % of tiles that are recalculated at a higher resolution). Thus, as we increased the recalculation rate, each tile has a higher probability of been recomputed at the second resolution level.

In Figure 5(a), we show the application execution time for each machine, using either CPU or GPU in each of them. The maximum and average standard deviation in these experiments are, respectively, 3.2% and 2.2%. The CPU execution

times, as presented, are strongly affected by the recalculation rate, as it impacts on the number of tiles processed at higher resolution, which are inefficient on the CPU.

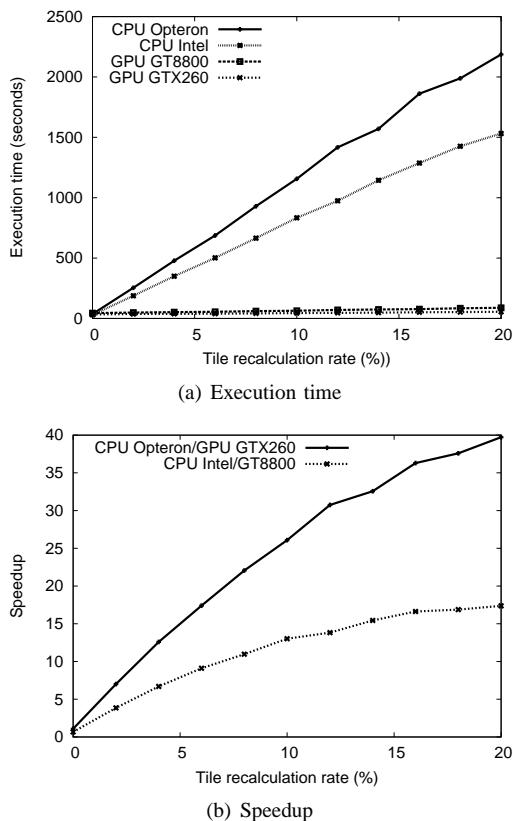


Fig. 5. Application evaluation on the both machines.

When we compared the two available CPUs, the Intel shown a better performance, although their clocks are almost the same. The GPUs evaluation presented better scalability and execution times of the GTX260, as expected, because it represents a newer generation containing more stream processors, bandwidth etc.

In Figure 5(b), we also presented the speedup between CPU and GPU for each machine. As these machines have the slower CPU and faster GPU on the same machine, the relative speedup have higher differences. These results are interesting for two reasons: (1) it shows that we should not compare an application running on GPU using only the speedup relative to the CPU, because it can may be susceptible to variations. For instance, if we had the Intel CPU, which is the fastest, on the same node as the GTX260, the GPU relative speedup could be reduced from the measured 39.72 to 27.82, while the AMD CPU and the 8800GT together should in-

crease GPU relative performance in 42%; (2) these different relative performances are interesting to our evaluation, as we can test the effectiveness of our task scheduling under both scenarios. As we compare the improvements obtained by our scheduling in the same hardware configurations, we do not incur in the errors discussed before.

B. Heterogeneous Environment Analysis

In this section, we evaluate our approach to execute NBIA on heterogeneous environments using the hardware described earlier: (i) Intel Core 2 Duo CPU 2.13GHz/NVIDIA GeForce 8800GT machine; (ii) dual quad-core AMD Opteron 2.00GHz/NVIDIA GeForce GTX 260 machine.

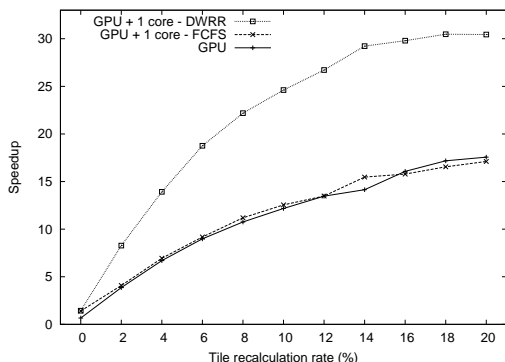


Fig. 6. Intel dual-core: CPU + GPU efficiency evaluation.

In Figure 6, we present the NBIA performance analysis, using the first hardware configuration, for multiple versions of the application: GPU-only, and GPU+CPU, collaboratively, with FCFS and DWRR scheduling. The standard deviation for these experiments were never bigger 4% for FCFS, and 3% for DWRR.

The FCFS showed significant improvement over the GPU-only when the recalculation rate was 0% (no recalculation). As the CPU and GPU have similar performance for low resolution tiles, when there is no recalculation, each tile will take the same time on either device, and since there are two devices, instead of one (as is the case on pure GPU version), a factor of 2 improvement was achieved, as expected. Table I presents the number of tasks processed by CPU using 0% of recalculation. We can see that FCFS and DWRR have almost the same scheduling, consequently, incurring in a similar performance.

When we increase the recalculation rate however, we see the DWRR policy still almost doubling the speedup in relation to pure GPU, while

Recalc (%)	0		12	
Resolution	Low	High	Low	High
1 CPU core - FCFS	16049	0	263	251
1 CPU core - DWRR	15978	0	21592	4

TABLE I
INTEL DUAL-CORE: #TILES PROCESSED BY CPU AT EACH RESOLUTION/SCHEDULING.

the FCFS achieves almost no improvement. For instance, in 12% of the recalculation, the GPU-only version of the application is 13.46 times faster than CPU, while using CPU and GPU together incurred into speedups of: 13.48 and 26.71, respectively, for FCFS and DWRR.

These experiments are further detailed in Table I, which shows the profile of the tasks processed by CPU, with 12% of recalculation rate, under each scheduling. As shown, using FCFS the NBIA processed few low resolution tiles using the CPU, for which it is faster. When the application employed the DWRR, the CPU processed more than 80% of the small tiles, while the GPU did most of the high resolution ones.

In Figure 7, we present the NBIA execution times for FCFS, and DWRR on the AMD dual quad-core machine. The execution times using FCFS, as shown in Figure 7(a), are reduced as we increase the number of CPU threads. The reduction is also higher for low recalculation rates, where there are more tiles suitable for CPU, and, consequently, the scheduling has less influence on the performance. Although we have gains of using CPU for FCFS, they are only proportional to the speedup between the CPU/GPU.

The NBIA execution times for the DWRR scheduling are shown in Figure 7(b). The DWRR results are better than FCFS, as we increased the recalculation rate, showing the efficiency of the scheduling. The speedups for FCFS and DWRR are also presented in Figures 8(a) and 8(b), respectively. It shows that while FCFS only slightly increases the application performance, DWRR almost doubles it.

# of CPU cores	FCFS		DWRR	
	Low	High	Low	High
1	637	58	10714	1
2	1177	133	15748	2
3	1925	173	18614	5
4	2090	219	18634	28
5	2872	286	20070	40
6	3819	393	20417	76
7	4726	478	20266	57

TABLE II
DUAL QUAD-CORE: #TILES PROCESSED BY CPU AT EACH RESOLUTION/SCHED., FOR RECALCULATION RATE OF 10%.

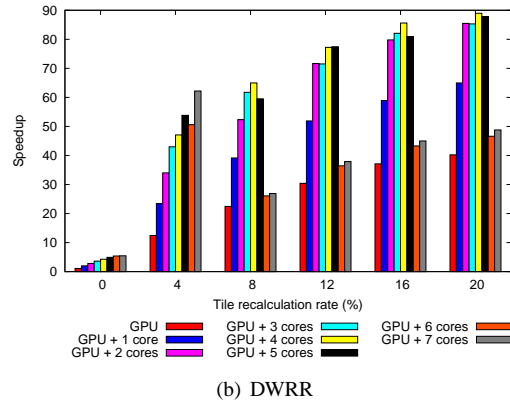
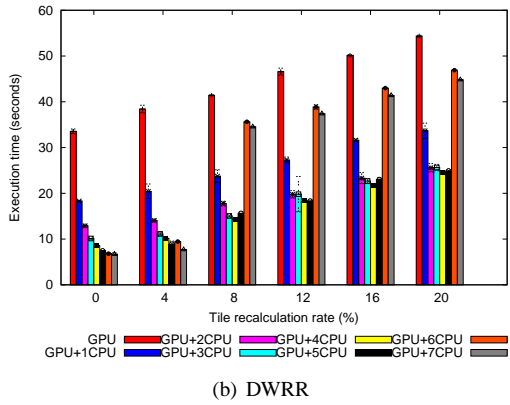
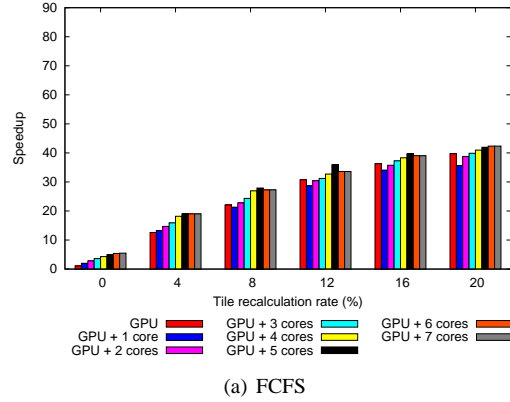
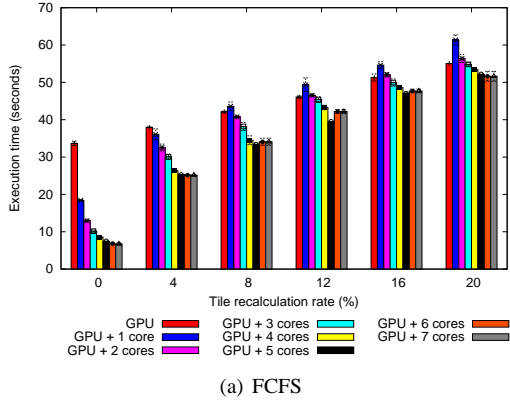


Fig. 7. Dual quad-core: NBIA execution time.

Fig. 8. Dual quad-core: NBIA CPU + GPU speedup.

The performance gap between FCFS and DWRR can also be explained through the tasks profile processed by each device. Table II presents the number of tiles processed by CPU, as we increased the number of CPU cores, maintaining 10% of recalculation rate. The FCFS provided a scheduling where the number of low resolution tiles, which are suitable to CPU, processed by CPU cores are much smaller than DWRR, while the high resolution tiles have reverse behavior.

The DWRR analysis shows that the performance is degraded after 5 CPU-cores because CPU-threads start competing for the available cores. Although we used 5 CPU-cores as processing units, the 3 CPU-cores remaining are been used by the Reader filter, to managing the GPU, and by the Anthill process that is responsible to managing the communication.

C. Distributed Environment Evaluation

In order to evaluate the performance of our application in a distributed environment, we performed experiments using our two clusters and varied the NBIA versions as well as the scheduling strategy. These experiments were performed using one copy

of each filter per machine, and 26,742 tiles images, as before. We also varied the recalculation rate from 0% to 20%, but the results did not present substantial variation. Due to that, and to improve the legibility, we only show the results for a 10% rate.

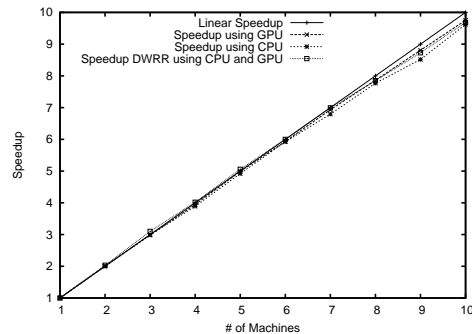


Fig. 9. Speedup: Intel Dual Core/8800GT cluster.

The speedup results into the Intel dual core cluster are shown in Figure 9. For these experiments, the speedups for all versions of the NBIA are close to linear as we increased the number of machines.

Finally, in Figure 10, we present the speedups for the AMD dual quad-core cluster. We see the same

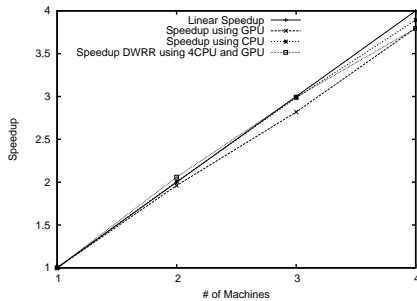


Fig. 10. Speedup: two Opteron Quad-core/GTX260 cluster.

near-linear behavior as before, as we scale the number of hosts on the run-time. The experiments for the DWRR were performed using 4 CPU cores and the GPU, as it is the most efficient configuration of this scheduling.

V. RELATED WORK

We have recently seen a growth in the number of projects which focus on developing frameworks for high performance computing models that are derived from the data-flow computing model, e.g., MapReduce [10], DataCutter [6], SPC [11], and Dryad [12]. These initiatives are mainly driven by the need to process a large amount of data from several scenarios (for instance, biomedical image analysis, and web-based information), and by the data-flow model inherent parallelism.

These systems have contributed significantly to the state-of-art, developing new important features. For example, DataCutter has created the transparent copies mechanism, which replicates processing stages providing data parallelism, and SPC which is able to dynamically add and manage streams. However, their focus is not on efficiently exploiting modern heterogeneous environments, like a cluster of CPUs and GPUs. In our approach, we try to maximize the application's throughput by using all the available resources efficiently, applying scheduling techniques to choose the device which is more suitable for handling a certain event.

Different works have also demonstrated the efficiency of using GPUs as coprocessors for a single node. Mars [13] is one of such systems. It is an implementation of the MapReduce programming model in the context of GPUs. Mars implements a subset of the MapReduce API in order to allow the development of MapReduce applications in such environments. Mars demonstrates its efficiency with speedups between 1.5 and 16 for six applications previously developed using Phoenix. Mars also performed a very initial investigation of the gains from using GPU and CPU cooperatively.

This evaluation was performed by statically dividing the tasks of the MapReduce stages statically according to the speedup between Phoenix and Mars, and processing each of the tasks' subsets in one of the systems. The Mars proposal is interesting, because it alleviates the hard task of developing applications to GPUs. Although it is a promising approach, this solution is still limited to a single machine node, and do not fulfill the original goal of massive distributed computing of the MapReduce programming framework [10].

The Merge framework [14] is another interesting programming model for heterogeneous environment, which, as Mars, proposed and evaluated the use of CPU and GPU as coprocessors, but it also is limited to a single process node. The results shown by both systems, Mars and Merge, when using GPU and CPU cores together, increased slightly the application performance, proportionally to the base speedup between these devices.

In this paper, we also use GPU, CPU, or both, collaboratively, but differently from Mars and Merge, our approach is not limited to a single machine, as we show in Section IV. In our work, we effectively exploit each device by scheduling tasks according to its performance on each available processor, which have been shown to be extremely efficient, as we could, for instance, in a dual-core machine, double the speedup of GPU-only version of our application adding a single CPU core.

The NBIA used to evaluate the scheduling techniques had been previously evaluated on GPU by Hartley et al. [15], where they also presented the first study of NBIA on cluster of GPUs machines. Their work also addressed the CUDA development process, including optimization issues, for NBIA. In this work, we implemented the multi-resolution NBIA using the Anthill framework to enable the use of GPU and CPU cooperatively, and showed that using heterogeneous resources efficiently can bring significant improvements to the performance, using the same source codes as presented in [15].

Finally, we have seen a considerable amount work on scheduling for heterogeneous environments. In [16], for instance, it is used a Greedy algorithm, which is classically employed for scheduling, to map tasks for parallel execution on heterogeneous environments based on an expected execution time. Other researches, like [17], have also investigated application level scheduling on distributed heterogeneous environments. Our work mainly differs from the previous ones regarding at least three points: (i) we investigate scheduling strategies for modern heterogeneous distributed en-

vironments; (ii) the proposed schedule considers a new scenario, where the relative performance between processor/machines is not fixed but varies according to the inputs of the tasks; (iii) we performed a filter level scheduling, where we try to optimize the execution of the application by increasing the performance of each filter.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an approach to use the CPU and GPU in a coordinated way in order to improve the efficiency of applications even further. We improved our event-driven Anthill run-time framework to incorporate a decision mechanism that analyses each event on a filter and decides whether the event should be executed on the GPU or the CPU and invokes the appropriate event handler. We create two simple scheduling policies for implement such decision mechanism: FCFS and DWRR.

We used a real world, compute intensive application of image analysis for classification of tumor images to evaluate our approach. Overall, our experiments have shown that substantial improvements can be made to the application performance. We have measured a factor of about 2 improvement on top of the GPU-only implementation of the applications. We also showed that the improvements is maintained linearly with the addition on multiple heterogeneous nodes to the execution environment.

As future work, we intend to evaluate the impact of different scheduling techniques to heterogeneous environments. The study of the two different schedulings in this paper gave us the insight on the importance of adequately matching events with the most profitable device, which can be very irregular and data-dependent. We are considering trying to assess it by creating a multi-level scheduling approach that combines the application filters placement, and the filter scheduling into heterogeneous devices employed on this paper.

REFERENCES

- [1] L. Teot, R. Khayat, S. Qualman, G. Reaman, and D. Parham, "The problems and promise of central pathology review: development of a standardized procedure for the children's oncology group," *Pediatric and Developmental Pathology*, vol. 10, no. 3, pp. 199–207, 2007.
- [2] O. Sertel, J. Kong, H. Shimada, U. V. Catalyurek, J. H. Saltz, and M. N. Gurcan, "Computer-aided prognosis of neuroblastoma on whole-slide images: Classification of stromal development," *Pattern Recognition, Special Issue on Digital Image Processing and Pattern Recognition Techniques for the Detection of Cancer*, vol. 42, no. 6, pp. 1093–1103, 2009.
- [3] H. Shimada, I. M. Ambros, L. P. Dehner, J. I. Hata, V. V. Joshi, and B. Roald, "Terminology and morphologic criteria of neuroblastic tumors: recommendation by the international neuroblastoma pathology committee," *Cancer*, vol. 86, no. 2, pp. 349–363, 1999.
- [4] G. Teodoro, D. Fireman, D. Guedes, W. M. Jr., and R. Ferreira, "Achieving multi-level parallelism in the filter-labeled stream programming model," *Parallel Processing, International Conference on*, vol. 0, pp. 287–294, 2008.
- [5] R. Ferreira, W. M. Jr., D. Guedes, L. Drummond, B. Coutinho, G. Teodoro, T. Tavares, R. Araujo, and G. Ferreira, "Anthill: a scalable run-time environment for data mining applications," in *Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD)*, 2005.
- [6] M. Beynon, R. Ferreira, T. M. Kurc, A. Sussman, and J. H. Saltz, "DataCutter: Middleware for filtering very large scientific datasets on archival storage systems," in *IEEE Symposium on Mass Storage Systems*, 2000, pp. 119–134. [Online]. Available: citeseer.ist.psu.edu/beynon00datacutter.html
- [7] S. W. O'Malley and L. L. Peterson, "A dynamic network architecture," *ACM Trans. Comput. Syst.*, vol. 10, no. 2, pp. 110–143, 1992.
- [8] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu, "Coyote: a system for constructing fine-grain configurable communication services," *ACM Trans. Comput. Syst.*, vol. 16, no. 4, pp. 321–366, 1998.
- [9] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multi-resolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 24, pp. 971–987, 2002.
- [10] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters." in *Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, December 2004, pp. 137–150.
- [11] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, P. Selo, Y. Park, and C. Venkatramani, "Sp: a distributed, scalable platform for data mining," in *DMSSP '06: Proceedings of the 4th international workshop on Data mining standards, services and platforms*. New York, NY, USA: ACM, 2006, pp. 27–37.
- [12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. New York, NY, USA: ACM, 2007, pp. 59–72.
- [13] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A mapreduce framework on graphics processors," in *Parallel Architectures and Compilation Techniques*, 2008.
- [14] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: a programming model for heterogeneous multi-core systems," *SIGPLAN Not.*, vol. 43, no. 3, pp. 287–296, 2008.
- [15] T. D. Hartley, U. V. Catalyurek, A. Ruiz, M. Ujaldon, F. Igual, and R. Mayo, "Biomedical image analysis on a cooperative cluster of gpus and multicores," in *22nd ACM Intl. Conference on Supercomputing*, Dec 2008.
- [16] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *HCW '98: Proceedings of the Seventh Heterogeneous Computing Workshop*. Washington, DC, USA: IEEE Computer Society, 1998, p. 79.
- [17] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-level scheduling on distributed heterogeneous networks," in *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 1996, p. 39.