SCHEDULING ALGORITHM FOR CPU/GPU NODE IN DISTRIBUTED COMPUTING

Suman Goyat[1], A.K. Soni[2]

[1]PhD Student, [2]Professor, Department of Computer Science & Engineering,
Sharda University, Greater Noida, UP.(India)

ABSTRACT

Over the past two decades, advancements in microelectronic technology have resulted in the availability of fast, inexpensive processors, and advancements in communication technology have resulted in the availability of cost effective and highly efficient computer method for dynamically scheduling applications running on heterogeneous platforms in order to maximize overall throughput. The key to our approach is accurately estimating when an application would finish execution on a given device based on historical runtime information, allowing us to networks. An essential component of effective use of distributed systems is proper task placement or scheduling. To produce high-quality schedules, scheduling algorithms require underlying support mechanisms that provide information describing the distributed system. We present a make scheduling decisions that are both globally and locally efficient. We evaluate our approach with a set of applications running on a system with a multi-core CPU and a discrete GPU.

Keywords: Distributed Systems; Cluster; Grid Computing; Grid Scheduling; Workload Modeling; Performance Evaluation; Simulation; Load Balancing; Task Synchronization; Parallel Processing.

I. INTRODUCTION

A distributed system is a collection of cooperating computers. In the past two decades, the use of distributed systems has increased dramatically. Such systems have several advantages over uni-processors, such as improved performance and increased fault tolerance. Nowadays, it is feasible to build computer systems with enormous processing capacities by interconnecting many small computers. An example of such a high-performance system is the Distributed ASCI (Advanced School for Computing and Imaging) Supercomputer [17], a set of four clusters of workstations at Dutch universities interconnected by Asynchronous Transfer Mode (ATM) links. Many compute-intensive applications, such as those found in the areas of weather forecasting, VLSI design, and other numerical computations, can benefit from the capacity of such super clusters. The features of distributed system are as follows:

- Data is stored in several sites (nodes), geographically or administratively across multiple systems
- Each site is running as an independent system
- What do we get?
  - Increased availability and reliability
  - Increased parallelism

Complications of distributed system are:

- Catalogue management: distributed data independence and distributed transaction atomicity

- Query processing and optimization: replication and fragmentation
- Increased update costs, concurrency control: locking, deadlock, commit protocol, recovery.

Distributed computing consists of applications running on a platform that has more than one computational unit with different architectures, such as a multi-core CPU and a many-core GPU. Generally these kernels perform better on the GPU as they are optimized for a GPU's highly parallel architecture and GPUs typically provide higher peak throughput. Therefore, applications preferentially schedule kernels on GPUs, leading to device contention and limiting overall throughput. In some cases, a better scheduling decision runs some kernels on the CPU, and even though they take longer than they would if run on the GPU, they still finish faster than if they were to wait for the GPU to be free. We propose that by capturing and using historical runtime information, a scheduling algorithm is able to make a decision about the tradeoff of forcing an application to run its kernel on a slower device against waiting for the faster device to become available.

Kernels perform better on the GPU as they are optimized for a GPU's highly parallel architecture and GPUs typically provide higher peak throughput. A better scheduling decision runs some kernels on the CPU as they can finish faster than if they were to wait for the GPU to be free. Utilizing all available processors for computational work, the total throughput of the system is increased over a static schedule that runs each kernel on the fastest device.

Dynamic approach used to heterogeneous scheduling is to predict how long an application will run when it is assigned. Capturing and using historical runtime information, a scheduling algorithm is able to make a decision about to run its kernel on a slower device against waiting for the faster device to become available. Implement the scheduler on a set of few applications and demonstrate the improvement of the algorithm (Dynamic Heterogeneous Scheduling Decisions Using Historical Runtime Data) over other scheduling decisions for a system that has a CPU and a GPU. We propose that by capturing and using historical runtime information, a scheduling algorithm is able to make a decision about the tradeoff of forcing an application to run its kernel on a slower device against waiting for the faster device to become available [11]. As a database of application runtimes gets built, scheduling decisions become better. In this work, we show that the database can be as simple as keeping an average runtime for each application, along with information about the input data size. From this information we also show that a dynamic scheduler can improve overall throughput considerably over a statically scheduled solution.

1.1 Details of Work

- We present an algorithm that analyzes a queue of applications and assigns them to devices of a heterogeneous system as overall computational throughput is increased over a statically scheduled solution.
- We demonstrate that even if all applications natively run faster when assigned to one device, there are situations where assigning an application to a slower device allows that application to complete before it would have if it had waited for the faster device.
- Furthermore, this solution preserves fairness for an individual's placement in the queue.
- A better scheduling decision runs some kernels on the CPU as they can finish faster than if they were to wait for the GPU to be free.
- We describe a history database structure that collects and averages runtime data for each application.
- We demonstrate how runtime predictions for applications with unique data inputs can be made from data input size differences.

- We implement the scheduler on a set of five Open CL applications and demonstrate the improvement of the algorithm over other scheduling decisions for a system that has a CPU and a GPU.
- We also show that the scheduler produces an improved schedule and a high utilization for both devices even when all applications individually run faster when assigned to the GPU.

II. PROBLEM DEFINATION

There are two types of distributed systems:

- Homogeneous system: identical DBMS, aware of each other, cooperate
- Heterogeneous system: different schemas/DBMS
- ❖ Multidatabase system: uniform logical view of the data -- common schema
- ❖ Difficult, yet common: system is typically gradually developed

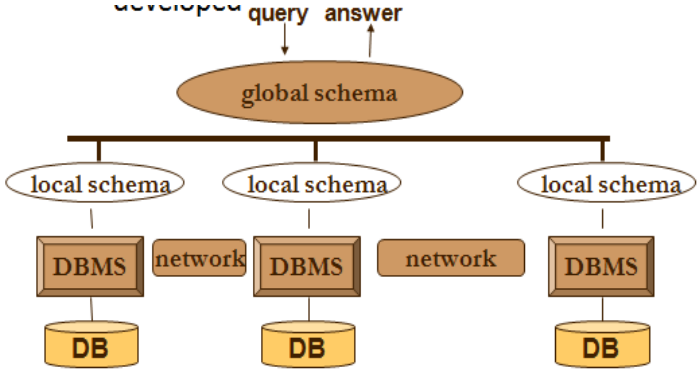


Figure 1 Architecture of Distributed System

Scheduling computational work for heterogeneous computer systems is substantially different than scheduling for systems with homogenous processing cores. The major differences of heterogeneous system are:

- ❖ An application can have a drastically different running time when assigned to different device: As GPU kernels running one hundred times faster than comparable CPU kernels and because of these runtime differences, assigning an application to one of two devices necessitates knowing which device will allow the application to run faster.
- ❖ GPU processors do not have the capability to time-slice workloads: Kernels that are launched on a GPU run sequentially, one at a time. The latest GPUs have limited ability for multiple kernels to run in parallel, but there must be careful coordination to ensure that all kernels and their data fit onto the card, and that they do not have any dependencies. Without the ability to time-slice, a kernel launched behind other kernels must wait until all other kernels finish completely before starting its own work.

We propose that scheduler determines the best device at a given time for each kernel by analyzing predicted runtimes of the applications. This scheduler has historical runtime information about the other applications in the queue, and knows which kernels, if any, are currently running. Given a set of queued applications that are not queued for a specific device, the scheduling problem becomes one of judiciously assigning the applications to devices to maximize computational throughput while remaining fair to the queue order. Many factors can go into this scheduling decision for a given application, including the number of applications ahead in the queue, the application or applications that are currently assigned to the devices on the system and their runtimes, how much data must be transferred between device memory systems, and the relative speed of the application when assigned to each device in the system.

Our implementation supports running the same kernel across both CPUs and GPUs. Kernels can be compiled for available devices prior to or at runtime. Kernels can be run on more than one device in a system. Our scheme allows for implementations that have separate versions of a kernel for each available device and the runtime similarly chooses the correct binary once a device decision has been made. We address occurring problems as difference between homogenous and heterogeneous by using a historical database that contains runtimes for applications on each device and determining a schedule that runs applications on the device in which they will finish first.

Problem can be defined as:

- Assigning an application to one of two devices necessitates knowing which device will allow the application to run faster.
- Assign applications to devices to maximize computational throughput while remaining fair to the queue order.
- Running the same kernel across both CPUs and GPUs.
- Allow for implementations that have separate versions of a kernel for each available device.
- Historical database that contains runtimes for applications on each device and determining a schedule that runs applications on the device in which they will finish first.

III. CREATING HISTORICAL DATABASE

Our method for dynamic scheduling relies on historical data about application runtimes. We propose a method for collecting data such that it is accessible quickly and provides enough information about a given application to be useful for making predictions about how long an application will take when assigned to a device. Table 1 shows the lightweight and extensible data structure we use to store the data. The data structure presented in Table 1 is not exhaustive, but we believe that in this form it is both robust enough to provide worthwhile data and lightweight enough to be useful for fast access. Given database provide CPU burst time and GPU burst time of each application. It will help to decide which application can be executed on CPU or GPU or alternatively.

Table 1 Historical Database

Job no	Value	Application	CPU Burst time	GPU Burst time
1	Float	MatTrp	8.00	15.00
2	Float	MatMul	22.00	13.00
3	Float	F_F_T	21.00	13.00
4	Float	EIGEN	21.00	14.00
5	Float	BinSrc	7.00	17.00

IV. LITERATURE SURVEY

The scheduling decisions are based on the dynamic parameters that may change during run time. The goal of scheduling is to utilize all the processors with minimum execution time by proper allocation of tasks to the processors. Task scheduling achieves high performance in the heterogeneous systems. A Parallel application can be represented by a Directed Acyclic Graph (DAG), which represents the dependency among tasks, based on

their execution time and communication time. We investigate different aspects in scheduling and issues in various levels of the heterogeneous systems.

The idea of using historical data for heterogeneous scheduling decisions has been discussed in other work, although very few have targeted GPU computing. Several researchers have proposed using performance models to predict runtime (e.g., Meng and Skadron [6] and Hong et al. [4]), but performance models have high overhead and are generally not portable between hardware generations. We believe that using historical runtime data provides a better prediction for kernel runtimes. Ali et al. show that using a historical prediction database is worthwhile for grid computing [13], and Siegel et al. [16] discuss automated heterogeneous scheduling where one stage is to profile tasks and to perform analytical benchmarking of individual tasks. This information is then used in a later stage to predict runtimes for applications based on current processor loads. Similarly, our approach profiles applications as they run, but also extrapolates runtimes for applications with different input sizes from an analytical assessment of the collected data. Topcuoglu et al. describe the "Heterogeneous Earliest-Finish-Time" (HEFT) algorithm [15], which, like our approach, attempts to minimize when individual applications finish. Maheswaran et al. [14] describe a set of heuristics that inform a dynamic heterogeneous scheduler. Their Min-min heuristic calculates which device will provide the earliest expected completion time across a set of tasks on a system. The task in the queue that will complete first is scheduled next. Both Topcuoglu et al. and Maheswaran et al. were written prior to the advent of GPU computing, and they simulated their algorithms. Our approach differs from both Topcuoglu et al. and Maheswaran et al. by considering fairness, ensuring that applications do not get pre-empted by other applications, and we also tested our scheduler on CPU/GPU heterogeneous hardware.

Harmony [12] also uses performance estimates in order to schedule applications across a heterogeneous system. They propose an online monitoring of kernels and describe a dependence-driven scheduling that analyzes how applications share data and decides on processor allocation based on which applications can run without blocking. Our approach considers applications to be independent, and schedules applications from multiple applications concurrently. Jim´enez et al. [7] demonstrate two predictive algorithms that use performance history to schedule a queue of applications between a CPU and a GPU: history-gpu, that schedules work on the first available device that can run an application, and estimate-hist, that estimates the waiting time for each device and schedules an application to the device that will be free the soonest.

Luk et al. [8] use a historical database for Qilin that holds runtime data for applications it has seen before, although Qilin focuses on breaking a single applicationacross multiple devices instead of running multiple applications across multiple devices as we do. Augonnet et al. [9] use performance models to provide scheduling hints for their StarPU scheduler, and programmers who write applications for StarPU can provide a "cost model" for each application that enables the scheduler to predict relative runtimes. Our approach does not require programmers to modify their code. Becci etal. [2] use performance and data transfer overhead history to inform a dynamic heterogeneous scheduler for legacy kernels, focusing on postponing data transfer between devices until it is actually needed. Research Gaps

Table 2

Author's names	Year of publication	Techniques used	Description
Delgado Peris; Hernandez,J.M.; Huedo, E.	2014	Pull based Late binding overlays	Job agents are submitted to resources with the duty of retrieving real workload from a central queue at runtime[3]
Jianlong zong; Bingshang He	2014	Kernelet: High-Throughput GPU Kernel Executions with Dynamic Slicing and Scheduling	Many kernels are submitted to GPUs from different users, and throughput is an important metric for performance and total ownership cost
Fengguang Song; Jack Dongarra	2012	Concurrent Programming - Parallel programming	A Scalable Framework for Heterogeneous GPU-Based Clusters: Devised a distributed dynamic scheduling runtime system to schedule tasks, and transfer data between hybrid CPU-GPU compute nodes transparently.
Tarun Beri, Sorav Bansal and Subodh Kumar	2011	Shared global address space, made efficient by transaction style bulk-synchronous semantics.	A runtime system for simple and efficient programming of CPU+GPU clusters. The programmer focuses on core logic, while the system undertakes task allocation, load balancing, scheduling, data transfer, etc

Tarun Beri, Sorav Bansal and Subodh Kumar[1] present a runtime system for simple and efficient programming of CPU+GPU clusters. The programmer focuses on core logic, while the system undertakes task allocation, load balancing, scheduling, data transfer

V. RESEARCH METHODOLOGY

We assume that applications are placed into a first-in-first-out (FIFO) queue and each kernel can run on any of the available devices. For clarity we also assume that there are two devices available, a CPU and a GPU, although the algorithm could easily be extended to include an arbitrary number of devices. We also assume that most applications will run faster when assigned to the GPU.

5.1 Steps of the Algorithm

In essence, the scheduler we describe implements a greedy algorithm that assigns applications to devices based on a comparison between the predicted times for the application to finish on all available devices. Even if an

application would run faster assigned to a particular device, if there are enough applications ahead of it in the queue for that device, it may finish faster assigned to the slower device because that device is free. Our scheduling algorithm is laid out as follows. There are two devices available most applications will run faster when assigned to the GPU. We create a sub-queue for each device, and place applications in those sub-queues from the main queue according to the following rules:

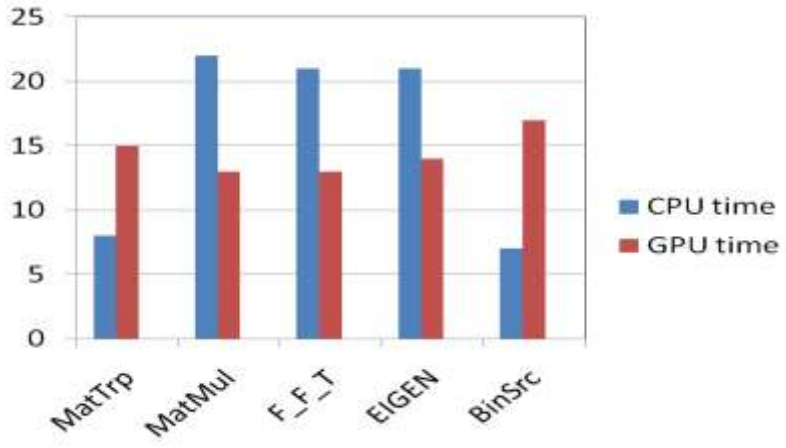
- If a sub queue has applications and the device for that sub-queue becomes free, assign the next application in the sub-queue on that device.
- If one device is busy but the other device is free and the next application in the main queue has not been assigned to that device before, assign it to that device in order to build the database.
- If only one device is free and the next application in the main queue runs slower assigned to that device, estimate how long the other device will be busy using the history database and include other application also scheduled to be assigned to that device in its sub-queue.
- If the next application in the main queue will finish faster by being assigned to the slower device, assign it to that device. If not, put it into the sub-queue for the busy device.
- Continue through the main queue until both devices are busy running applications.

As an application finishes, update the historical database with the runtime information, calculating the average runtime and the standard deviation, and repeat the algorithm from the beginning. The scheduling algorithm described above continues to improve as more data is entered into the historical database, and each application is penalized at most once when it is assigned to a slower device in order to build the database. Because application runtimes are averaged into the previous runtime for a device, outlying points that could be caused by factors unknown to the scheduler (e.g., GPU contention due to video processing associated with the display) are smoothed out over time. Because our scheduler assigns applications to devices that are not necessary optimal for each individual application, we must discuss scheduling fairness. We define a schedule to be fair if each application finishes no later than it would have finished if it were allowed to execute its kernel on its preferred device. In other words, a fair schedule does not penalize an application even if the application is assigned to its non-preferred device. Accordingly, no starvation occurs, and applications are scheduled for a device in queue order. The algorithm presented earlier generates fair schedules except in two cases: if the predicted runtimes are significantly incorrect or when an application is first encountered and is assigned to a slower device.

5.2 Application Runtime Prediction

If an application has been assigned to a device at least once with a given set of inputs and the same application is subsequently run with the same input size, the scheduler uses the average application time as a runtime prediction for that device. The scheduler makes the prediction based on a linear least-squares calculation, using the input sizes as one parameter, and the previous runtimes as another. Table shows the actual runtimes versus the predicted runtimes.

Table 3 CPU and GPU utilization of 5 applications



IV. WORKLOAD AND TEST ENVIRONMENT

In order to test our algorithm, we used five applications with one kernel, and ran the set of applications sequentially. The applications we used in our experiments represent a number of computational algorithms that are commonly used in scientific computing. We will get the applications and the absolute and relative runtimes for the data sets that we tested with. As expected, most applications had kernel that ran faster on the GPU, and therefore the entire applications ran faster on the GPU. In order to demonstrate the scheduler when some application were faster assigned to the CPU, we set the data size small enough for these applications.

Initially, with limited or no historical information, applications are assigned to the GPU if it is free and to the CPU otherwise. Compared to a GPU-only scheduling solution, the scheduler performs worse on first few runs, but it quickly improves its performance. The scheduler always performs better than a CPU-only solution for this set of applications.

VII. RESULTS

Results show the time needed to run the set of 5 applications for four different scheduling algorithms. In the CPU only and GPU only cases, all applications were assigned to each respective device. For the "CPU/GPU Static using history" case, each application is assigned to its preferred device. The last column shows the results using the dynamic scheduler we have described. The utilization of the CPU is only 37%, and most of the applications were assigned to the GPU. The results from our algorithm, demonstrating 67% GPU utilization.. We have put our data set for 5 applications into a C program and used our MIXED Runtime Scheduling Technique. Results obtained were :

Job No.	Application	CPU-Burst Time	GPU-Burst Time
1,	**MatTrp**	**8.00**	**15.00**
2.	**MatMul**	**22.00**	**13.00**
3.	**F_F_T**	**21.00**	**13.00**
4.	**EIGEN**	**21.00**	**14.00**
5.	**BinSrc**	**7.00**	**17.00**

The jobs were executed based on the minimum time it takes on a particular CPU/GPU.

Total CPU elapsed time was = 15.00 (Here jobs 1 and 5 were run on CPU and others on GPU)

Total GPU elapsed time was = 40.00

After mixing up the jobs, they were executed based on 1st priority on CPU/GPU where it takes less time and within it, in case a particular device is free the job ran on that device though it runs slow on it. Results were:

Total CPU elapsed time was = 28.00 (Job 1&5 of CPU-Burst Time and Job 3 of GPU-Burst Time)

Total GPU elapsed time was = 27.00 (Job 2&4 of GPU-Burst Time)

In 1st case CPU Utilization was: 37% (=15 x 100 / 40) In 1st case GPU Utilization was : 63%

In 1st case jobs finished in 40 time units.

In 2nd case CPU Utilization was : 100% (=28x100/28) In 2nd case GPU Utilization was : 96% (=27x100/ 28)

In 2nd case jobs finished in 28 time units.

A net saving of (40-28)*100/40=30% time units.

VIII. CONCLUSION

The dynamic scheduling finishes all applications **30% faster** than the statically scheduled GPU-only solution. In this paper, we described and demonstrated a dynamic scheduling algorithm that schedules application based on a historical database of runtime values. We showed that by storing and using the historical information, a scheduler can determine how to assign applications to processors. The resulting schedule fairly schedules applications according to their order in the queue, and if the runtime prediction is relatively accurate, applications will finish running prior to when they would have if they had all been statically scheduled onto the GPU, or if they had been scheduled to run on the device on which they run fastest.

IX. FUTURE WORK

We will take a look at how a historical scheduler could be used in a cluster of CPU/GPU machines, and for other heterogeneous machines including Cell/B.E. or embedded system. Developing such an algorithm is still an open problem. Graphic processors (GPUs) will be introduced with many light-weight data-parallel cores, will provide substantial parallel computational [1] power to accelerate general purpose applications. To best utilize the GPU's parallel computing resources, it is crucial to understand how GPU architectures and programming models can be applied to different categories of traditionally CPU applications[3].We will also describe methods for looking at runtime trends to predict runtimes for applications with unique data size inputs. We will illustrate that using a historical prediction database is worthwhile for grid computing [4].Multi level dynamic scheduling and multi heuristic task allocation approaches to be implemented. We will investigate scheduling and runtime framework for a cluster of heterogeneous machines [5].

REFRENCES

[1]. A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures, Commun. Comput. Phys. Vol. 15, No. 2, pp. 285-329 February 2014

[2]. M. Becchi, S. Byna, S. Cadambi, and S. Chakradhar. Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory. In ACM Symposium on Parallelism in Algorithms and Architectures, pages 82–91, 2010.

[3]. J. Gummaraju, L. Morichetti, M. Houston, B. Sander, B. R. Gaster, and B. Zheng. Twin peaks: a software platform for heterogeneous computing on general-purpose and graphics processors. In 19th Conference on Parallel Architectures and Compilation Techniques, pages 205–216, Vienna, Austria, 2010.

[4]. S. Hong and H. Kim. An integrated gpu power and performance model. In Proceedings of the 37th annual international symposium on Computer architecture, ISCA ’10, pages 280–289, New York, NY, USA, 2010.

[5]. V. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, et al. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In 37th International Symposium on Computer Architecture, pages 451–460, 2010.

[6]. J. Meng and K. Skadron. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In Proceedings of the 23rd international conference on Supercomputing, ICS ’09, pages 256–265, 2009.

[7]. V. J. Jim´enez, L. Vilanova, I. Gelado, M. Gil, G. Fursin, and N. Navarro. Predictive runtime code scheduling for heterogeneous architectures. In 4th Conference on High Performance Embedded Architectures and Compilers, pages 19–33, 2009.

[8]. C.-K. Luk, S. Hong, and H. Kim. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 45 –55, 2009.

[9]. C. Augonnet, S. Thibault, R. Namyst, and P. Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. In Euro-Par Conference on Parallel Processing, pages 863–874, 2009.

[10]. J. Meng and K. Skadron. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In Proceedings of the 23rd international conference on Supercomputing, ICS ’09, pages 256–265, 2009.

[11]. S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, and K. Skadron. A performance study of general-purpose applications on graphics processors using CUDA. Journal of Parallel and Distributed Computing, 68(10):1370–1380, 2008.

[12]. G. F. Diamos and S. Yalamanchili. Harmony: an execution model and runtime for heterogeneous many core systems. In 17th International Symposium on High Performance Distributed Computing, pages 197–200, Boston, MA, 2008.

[13]. Ali, A. Anjum, J. Bunn, R. Cavanaugh, F. van Lingen, R. McClatchey, M. A. Mehmood, H. Newman, C. Steenberg, M. Thomas, and I. Willers. Predicting the resource requirements of a job submission. In Computing in High Energy Physics, pages 130–134, Inter laken, Switzerland, September 2004.

[14]. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems page 30, Los Alamitos, CA, 1999.

[15]. H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In 8th Heterogeneous Computing Workshop, pages 3–14, 1999.

[16]. K.G. Langendoen. Myrinet: The High-Speed Network for DAS. In Proc. 13th Ann. Conf. of the Advanced School for Computing and Imaging, page 259. Heijen, The Netherlands, 1997.

[17]. H. J. Siegel, H. G. Dietz, and J. K. Antonio. Software support for heterogeneous computing. ACM Computing Surveys, 28(1):237–239, 1996.