X?Name?X: Efficient Execution of Hierarchical Pipelines on Heterogeneous Platforms

ABSTRACT

abstract.

1. MOTIVATION/OBJECTIVE

The successful introduction of accelerators as general purpose processors is changing current High Performance Computing (HPC) systems, which are nowadays being deployed as heterogeneous CPU-GPU equipped platforms. Taking advantage of those solutions, however, still be a very hard programming task, mainly due to the lack of high level programming languages, and runtime frameworks to abstract the complex interactions among applications and heterogeneous systems. While some work has been done in order to execute simple kernel applications using CPUs and GPUs cooperatively, including MapReduce computations for shared memory machines [5, 9, 10], the execution of more complex real world pipelined applications (e.g. with dependency among stages) in distributed environments still a challenging open problem. The goal of this work is to propose and implement runtime support to make easier/viable the deployment of real complex applications on distributed environments. To accomplish this tasks we will automate/hide from the user most of the complex tasks regarding interactions of the application with the executing platform, and provide a set of non-trivial optimizations/scheduling mechanisms.

2. RUNTIME SYSTEM OVERVIEW

Briefly, the runtime system we are building will execute dataflow applications with dependency among stages, utilizing distributed CPU-GPU equipped platforms. The overall execution model is based on a bag of tasks, where tasks are instantiations of a given stage of the application dataflow — the pair input data and stage processing functions. To guarantee correct ordering in the execution of such tasks, the runtime system will use task dependency information given as input by the user through a simple API, and will assert that a tasks is not dispatched for execution before all dependencies are solved.

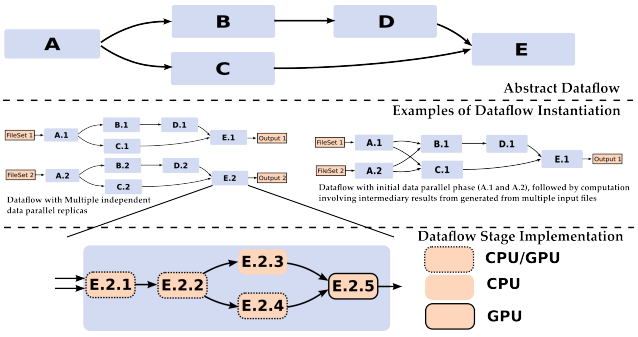

Figure 1: Sample Application Dataflow.

Further, communication among the pipeline stages, or tasks, is performed through a distributed file system, meaning that data is read and written to files between stages. Figure 1 presents a schema of the multiple levels that may be used to represent a dataflow in this model. In the first level, Abstract Dataflow, we simply have the logical stages of the application, describing their connections. In the second level, there is the instantiation of the dataflow, where the logical computing stages are associated to different input data, and dependencies among instantiation of stages (Tasks) are described. For instance, in the left side, there is an instantiation of the dataflow with an independent replication of the entire pipeline for each input data. In the right side, however, there is a more complex interaction in the dataflow as each input data is computed in parallel in through multiple instantiations of first stage, but the following stages on the pipeline will use results computed from the previous instantiations of stage A (A.1 and A.2). On the bottom level of the same figure, we show that each logical stage of the application may again be another pipeline, with dependencies among sub-stages that may additionally have implementation for multiple devices, e.g. CPU or/and GPU.

Providing a stage with the capabilities of being described as a pipeline of multiple sub-tasks has to be (i) with the need of exporting operations with different performance to the local scheduler (described latter, but similar to IPDPS), and (ii) with necessity of providing an efficient way for the application to execute a number of sub-stages without having to write the output of each of them to the file system,

XXX 2012 City

as is necessary in the communication among stages of the application pipeline.

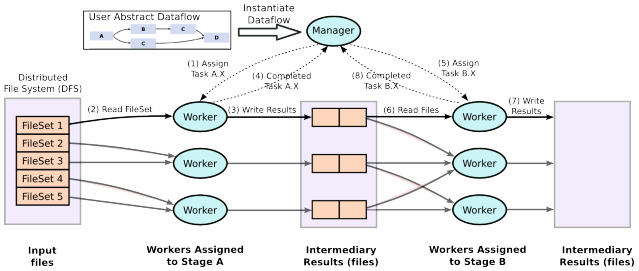

Figure 2: Overview of the system architecture, and task mapping.

Figure 2 shows the runtime system design, with an overview of the interaction among its components. In this system, the *Manager* is the process responsible for handling dependencies among different instantiations of the dataflow stages (Tasks) (middle level of Figure 1), and to dispatch those tasks for execution with the *Workers*. The *Workers* will communicate with the *Manager* to request tasks to compute, retrieve those tasks, and executed them. Each task will read data from a file system and may spawn a pipeline of sub-tasks that is locally scheduled by each *Worker*. When all sub-tasks of a given task are computed, the *Worker* informs the *Manager*, which may assign another task for execution. In practice, a single worker may execute multiple Stages, even concurrently, and both sets of workers presented in Figure 2 are not necessarily disjoint.

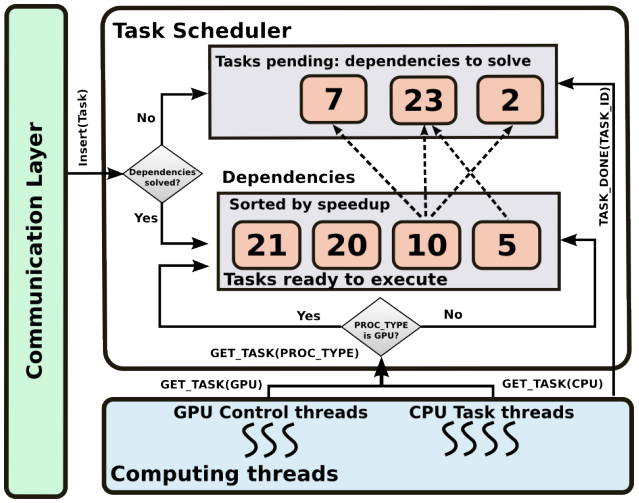

Figure 3: Workers Environment: the workers is a multi-thread process that may take advantage of all processing elements in a node to execute the applications' tasks.

The environment of each Worker, shown in Figure 3, is similar to what we proposed for IPDPS, with addition of a few features. As discussed, task dependency management has been added, as well as the communication layer module that is coupled to abstract exchange of information with the *Manager*.

3. RUNTIME SCHEDULING

In addition to building the runtime system infra-structure described in last section, it will be necessary to provide a hierarchical scheduling to assign task from the *Manager* to the *Workers*, called Manager Level Scheduling, as well as internally to each *Worker* — Worker Level Scheduling.

3.1 Manager Level Scheduling

The main objective is to assign tasks to *Workers* in order to optimize the performance of the overall system. There are two major aspects that should be considered: (i) Load imbalance among *Workers*, and a demand-driven task assignment among Workers and Manager should be enough for solving this problem; (ii) A smarter choice of which task to assign for a Worker could also potentially improve the performance. For instance, if a Worker already has a number of tasks with low speedup, it would be better to sent a task with high speedup for that Worker, which may be able to make better local scheduling decisions. This tends to be better than traditional FCFS scheduling, as we observed in the PRIORITY compared to FCFS. Below is the list of features that should be added to the Manager scheduler.

- Demand-driven tasks assignment;
- Multi-task assignment to overlap communication/computation;
- Performance Aware task choice;

3.2 Worker Level Scheduling

The Worker level scheduler is more aligned with what we already proposed for the IPDPS paper, with addition of dependency among tasks, and potentially new scheduling techniques that take into account the number of dependencies when choosing a task to executed. The summary of features is:

- Support to dependency among sub-tasks;
- Scheduling policies similar to those described in [15];
- May include number of tasks dependencies into the scheduling decision;

3.3 Analytical Performance Model

- Set of tools to analytically estimate applications' performance. We already have a draft that we intended to use with the HPDC, before it turn into Morphological Reconstruction;

4. LEARNING TO SCHEDULE

In our work, we have so far asked the programmer to provides us with information about the expected performance (speedup) of tasks when executed in the GPU. The idea here is to start working towards an automation of the process learning the performance of application (stages of the application), and use this information to automatically schedule the application in CPU-GPU environments.

A few works have started addressing this problem on the context of modeling and estimation of GPU application's performance, most of them using history based strategy based on application profiling [1, 4, 2, 6, 7]. Most of these

works, however, fail miserably to provide a generic approach, and are very simplistic though they already show that doing such a profiling based performance estimation is promising. A related interesting work for multicores CPUs only is [11], where the authors take advantage of profile history to estimate the set of optimizations that would result into the best performance for an application. In my understanding, we could leverage some of their propositions in our work. Moreover, there is also some work on performance modeling [12, 13] which states that computing relative performance among devices (different processors or disks, etc) is easier than trying to estimate execution times. This is good news once our scheduling policies are based on speedups. Another good news is that the accuracy of the speedup estimates is not critical as long as the order of tasks is not affected, as we stated for IPDPS. Also, a estimation that separates tasks with small and large speedups should be enough to achieve good performance.

To accomplish this task we will need the following components.

- Knowledge Database: to store performance data collected;
- Performance fitting algorithms: to estimate performance from the history data stored at the Knowledge database;
- Automated task grouping: is one of the features that could be derived from this knowledge.

5. OTHER RELATED WORK

These are some frameworks we should mention in the related work [3, 8, 14]

6. REFERENCES

[1] C. Augonnet, S. Thibault, and R. Namyst. Automatic calibration of performance models on heterogeneous multicore architectures. In *Proceedings of the 2009 international conference on Parallel processing*, Euro-Par'09, pages 56–65, Berlin, Heidelberg, 2010. Springer-Verlag.

[2] A. Binotto, B. Pedras, M. Go£ andtz, A. Kuijper, C. Pereira, A. Stork, and D. Fellner. Effective dynamic scheduling on heterogeneous multi/manycore desktop platforms. In *Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2010 22nd International Symposium on*, pages 37 –42, oct. 2010.

[3] G. Bosilca, A. Bouteiller, T. Herault, P. Lemarinier, N. Saengpatsa, S. Tomov, and J. Dongarra. Performance portability of a gpu enabled factorization with the dague framework. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 395 –402, sept. 2011.

[4] C. Gregg, J. Brantley, and K. Hazelwood. Contention-aware scheduling of parallel code for heterogeneous systems. In *2nd USENIX Workshop on Hot Topics in Parallelism*, HotPar, Berkeley, CA, June 2010.

[5] 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.

[6] V. J. Jiménez, L. Vilanova, I. Gelado, M. Gil, G. Fursin, and N. Navarro. Predictive runtime code scheduling for heterogeneous architectures. In *Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*, HiPEAC '09, pages 19–33, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] M. Kicherer, R. Buchty, and W. Karl. Cost-aware function migration in heterogeneous systems. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 137–145, New York, NY, USA, 2011. ACM.

[8] F. Le Mentec, T. Gautier, and V. Danjean. The X-Kaapi's Application Programming Interface. Part I: Data Flow Programming. Rapport Technique RT-0418, INRIA, Dec. 2011.

[9] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng. Merge: a programming model for heterogeneous multi-core systems. *SIGPLAN Not.*, 43(3):287–296, 2008.

[10] C.-K. Luk, S. Hong, and H. Kim. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *42nd International Symposium on Microarchitecture (MICRO)*, 2009.

[11] L. Luo, Y. Chen, C. Wu, S. Long, and G. Fursin. Finding representative sets of optimizations for adaptive multiversioning applications. In *International Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion*, Paphos, Cyprus, Jan. 2009.

[12] M. Mesnier, M. Wachs, B. Salmon, and G. R. Ganger. Relative fitness models for storage. *SIGMETRICS Perform. Eval. Rev.*, 33(4):23–28, 2006.

[13] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Relative fitness modeling. *Commun. ACM*, 52(4):91–96, 2009.

[14] V. Ravi, W. Ma, D. Chiu, and G. Agrawal. Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. In *Proceedings of the 24th ACM International Conference on Supercomputing*, page 137£146. ACM, 2010.

[15] G. Teodoro, T. M. Kurc, T. Pan, L. A. Cooper, J. Kong, P. Widener, and J. H. Saltz. Accelerating Large Scale Image Analyses on Parallel, CPU-GPU Equipped Systems. In *26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2012.