

# Efficient Runtime Support for Intra-Node Acceleration of Data Parallel Loops with Different Communication Patterns

Gagan Agrawal, Ohio State University, agrawal@cse.ohio-state.edu

## 1. Introduction

---

It is well accepted that heterogeneity and increasing intra-node complexity will be key features of future exascale systems. Many supercomputers already have heterogeneous parallelism within each node, with the CPU-GPU nodes, and the emerging architectures that feature an on-chip integration of a GPU are also likely to be used for supercomputers.

To maintain (performance) portability and programmer productivity, we must develop mechanisms for mapping computation specified in a high-level language (e.g. a data parallel or a domain-specific language) to heterogeneous cores within such node. This requires a highly efficient runtime system, capable of dynamically scheduling the computations and managing communication between different cores as required for the loop, while paying attention to the complex memory hierarchy.

## 2. Current Work

---

In preparation for handling complexities in the anticipated exascale systems, we have been working with the current heterogeneous systems, including architectures with a *decoupled* (NVIDIA) GPU and the AMD Fusion CPU-GPU. In the near future, we will continue to evaluate our framework on these and other upcoming systems (like the Intel MIC architecture), while ensuring the scalability of the framework to handle larger number of cores and even higher diversity of resources.

As we discussed earlier, one crucial challenge is to exploit the combined processing power of different types of cores, e.g. CPU cores and GPU cores, for the same computation. Thus, our goal is to divide the work between CPU cores and one or more GPUs on each node. We prefer dynamic schemes, since relative performance of different processing units can vary across applications and input datasets. Suppose, initially, that we can view the computations we are targeting as a collection of *independent tasks*. This is not entirely true when communication is involved, but can serve as an initial starting point, before we consider communication associated with each specific pattern.

In recent work at Ohio State, we have developed an extensive framework for runtime partitioning of work between CPU cores and a decoupled GPU. Using this framework, we have demonstrated how generalized reductions [4], stencil computations [3] and irregular reductions [1] can be scaled by using both CPU and GPU. The main issue in this work has been to minimize the number of times a GPU kernel is invoked, while also ensuring that idle time of either of the resources is very low. Other challenges have been minimizing the amount of communication for stencil computations [3], and effective partitioning of irregular meshes so that runtime scheduling is facilitated [1]. Our work has shown that significant speedups are possible for all classes of computations by using the GPU and the multi-core CPU simultaneously, over the best case between GPU-only and CPU-only execution.

In our ongoing work, we have developed a runtime framework with the same goal for the AMD Fusion architecture. The details of this framework are quite different, as communication over PCI-Express is no longer required to execute on GPUs. However, there are many complex challenges related to memory hierarchy, data movement and keeping the overheads of scheduling low. Since the same set of issues are likely to arise in each node of the anticipated exascale systems, we believe that the synchronization-free scheduling framework we have developed for AMD Fusion also forms the basis for exascale runtime services.

**Synchronization Free Scheduling:** Initially, we assume that the work in a data parallel loop has been divided into independent tasks, and communication is somehow handled correctly. One approach could be to maintain a single *task queue*, which will be shared and accessed by all the thread blocks from different processing units, like the CPU and the GPU. Clearly, in such a case, locking will be needed on the task queue to ensure that the same task is not taken by two different blocks. This design is not desirable, both for AMD Fusion, and in emerging systems with even more cores within a node, for several reasons.

*Basic Ideas:* In view of our observations above, we have designed a novel scheduling framework, which we refer to as the synchronization-free scheduling framework. The framework is summarized in Figure 1. It is based on the popular master-slave model. A master thread is responsible for scheduling the tasks from a task pool. Once all the tasks have been assigned, the master thread also informs all the thread blocks that the entire computation is finished. The scheduling decisions are made by the *scheduling module*, where different algorithms can be implemented. Compared to the scheduling methods that have been used for CPU-GPU environments [4] the smallest scheduling unit in our new model is not based on devices, but instead, on thread blocks, i.e., all thread blocks, from the CPU or the GPU, are assigned their tasks independently. This scheduling strategy avoids the synchronization overhead inside each device. Besides the scheduling module, the other main component of the system is a *task interface pool*, which can be viewed as a communication interface between the master thread and the thread blocks. The master and slave interact as follows. If there are available tasks in the task pool, the

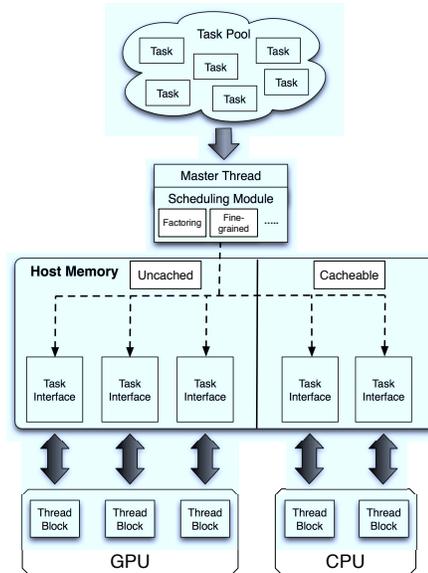
master thread will iterate all the existing thread blocks on all the devices, until it finds a *free* thread block. Next, the master thread will assign a new task to this thread block, the size of which depends upon the scheduling algorithm used. After it is assigned a new task, the thread block is informed that the new task is now available by updating a variable. On the slave side, one particular thread will be in charge of taking new tasks for all the threads from that thread block. If new work is available, this will store the starting point and the work size on the shared memory. Then, before doing the computation on the new task, it will also indicate to the master that it is ready to *prefetch* the next task. The benefit of prefetching is that the cost of distribution of the new tasks can be overlapped with the computation. From our discussion, we can see that no two threads update a particular task interface at any time, because of the way the conditions are used before reading or writing. This eliminates the need for any locking, and thus, the high overheads and even potential correctness problems.

*Observations - Evolving the Framework for Future Systems:*

Overall, our design can handle many challenges we anticipate in intra-node architectures of the anticipated exascale systems - massive concurrency, heterogeneity of resources, careful attention to memory hierarchy features, and careful load balancing while keeping the scheduling overheads negligible (as shown through preliminary results - not included here). Thus, our design can serve as a basic framework for intra-node scheduling for complex intra-node architectures, though specific location of task pools and granularity of scheduling may need to be adapted for each specific architecture. By customizing and optimizing the implementation for each architecture, we expect to support higher-level routines specific to each pattern, and thus, enable *performance portability*.

**Runtime Routines for Different Communication Patterns:**

Based on the scheduling framework described above (and the precursor framework designed for decoupled GPUs [1, 3, 4]) we have created runtime modules that can execute a loop with stencil computations, generalized reductions, or irregular reductions. For each pattern, we have developed runtime routine that can handle partitioning and communication, while invoking the task scheduling framework that has been customized for the particular architecture.



**Figure 1:** Synchronization Free Scheduling Framework for AMD Fusion

**3. Related Work**

For accelerating an application using CPU and GPU simultaneously, Teodoro *et al.* [5] describe a runtime framework that selects either of a CPU core or the GPU for a particular task. Recently, the Qilin system [2] has been developed with an adaptable scheme for mapping the computation between CPU and GPU simultaneously. The Qilin system trains a model for data distribution based on curve-fitting. Our work, in comparison, is based on dynamic work distribution, and does not require any offline training.

**4. Discussion**

**Challenges Addressed:** We are addressing the problem of scaling performance with increasing intra-node parallelism and heterogeneity, while maintaining portability and programmer productivity.

**Maturity:** The approach has been demonstrated to be successful in two existing class of architectures that involve intra-node heterogeneity.

**Uniqueness:** Architectures where a single node has a large number of (diverse) cores have not been available in the past - they are only beginning to emerge and will be the norm as we move towards the Exascale era. Thus, the framework we are proposing is quite unique to exascale computing.

**Novelty:** We are not aware of any other work which combines dynamic scheduling on heterogeneous cores with management of partitioning and communication. Most work has been on scheduling of independent tasks.

**Applicability:** The work will be broadly applicable towards supporting high-level APIs for parallel computations.

**Effort:** The effort will depend upon 1) how many diverse communication patterns arise in applications that need to be ported on exascale architectures, and 2) how many different architectures exist. We have been successfully developed solutions for three common patterns on two different architectures over the last 2 years.

## References

---

- [1] Xin Huo, Vignesh T. Ravi, and Gagan Agrawal. Porting irregular reductions on heterogeneous cpu-gpu configurations. In *HiPC'11*, Dec. 2011.
- [2] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Micro-42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 45–55, New York, NY, USA, 2009. ACM.
- [3] Vignesh T. Ravi and Gagan Agrawal. A dynamic scheduling framework for emerging heterogeneous systems. In *HiPC'11*, Dec. 2011.
- [4] Vignesh T. Ravi, Wenjing Ma, Vignesh T. Ravi, and Gagan Agrawal. Compiler and Runtime Support for Enabling Generalized Reduction Computations on Heterogeneous Parallel Configurations'. In *Proceedings of International Conference on Supercomputing (ICS)*, 2010.
- [5] George Teodoro, Timothy Hartley, Umit Catalyurek, and Renato Ferreira. Run-time Optimizations for Replicated Dataflows on Heterogeneous Environments. In *Proceedings of Conference on High Performance Distributed Computing (HPDC)*, June 2010.