# Hermes: Scalable and Load Distribution Engine for General-Purpose Computing on Graphics Processing Units (GPGPU)

Junghee Lee, Chrysostomos Nicopoulos*, Hyung Gyu Lee, Dongkun Shin† and Jongman Kim
*Georgia Institute of Technology, *University of Cyprus, † Sungkyunkwan University*
*{junghee.lee, hyunggyu, jkim}@gatech.edu, *nicopoulos.chrysostomos@ucy.ac.cy, † dongkun@skku.edu*

## Abstract

*Modern Graphics Processing Units (GPU) constitute typical examples of multicore systems with more than one hundred processing. However, in order to efficiently exploit the increasing number of on-chip processing cores, a scalable enabler of massively parallel computing is imperative. This paper proposes Hermes, a scalable and dynamic load distribution engine that exploits hardware aggressively to reinforce massively parallel computation in many-core settings. Our experimental results show that Hermes is scalable. Given the same amount of jobs, using Hermes is at least 4.5 times faster than existing techniques when the number of processing cores is 4096.*

**Keywords:** Load balancing, GPU, Many-core

## 1. Introduction

Modern Graphics Processing Units (GPUs) are typical examples of many-core systems with more than one hundred simple processing cores. Having recently been provided with extensive programming support, GPUs are becoming extremely powerful general-purpose hardware accelerators known as General-Purpose Computing on Graphics Processing Units (GPGPU).

Even though many applications running on GPU have regular computation kernels, the scheduling mechanism of the GPU itself and the different memory access patterns can cause load imbalance among the processing elements. Furthermore, there are many applications with an inherently irregular computation kernel – like the hierarchical radiosity method, and volume rendering [5] – which inevitably lead to load imbalance. The scalability of dynamic load distribution is critical for GPU applications. Recent research [1] has indicated that previously proposed parallel computing paradigms face scalability issues.

This paper proposes Hermes[1], a scalable, hardware-based, dynamic load distribution engine that enhances concurrency control and ensures uniform utilization of computational resources. This engine is overlaid on top of the existing GPU infrastructure, it is completely independent of the on-chip interconnection network, and it is transparent to the operation of the system.

## 2. Hermes

The programming model of GPU is similar to a fork-and-join model. The CPU passes a computation kernel, which is generally a function, with the number of threads to be created within the GPU. All the necessary arguments of the function should be provided at this time. The GPU subsequently creates all the threads and executes them on multiple processing elements concurrently. The CPU continues after the result is returned from GPU, once all the threads are finished. New threads cannot be created during execution time on the GPU (within the context of GPGPU). The number of threads should be determined at the time when the computation kernel is passed to the GPU.

Hermes consists of a number of load-balancing nodes (one such node for each processing element in the system), arranged as a mesh-based micro network overlaid on top of the existing GPU infrastructure. Note that Hermes is a distinct micro-network that is totally independent of any existing on-chip interconnection network. In other words, the load balancing mechanism does not interfere with the activities of the GPU interconnection backbone. This is in sharp contrast with Carbon's approach [2] that utilizes the same on-chip network as the cache sub-system.

[1] In Greek mythology, Hermes was the messenger of the gods. Much like this Olympian god, our micro-network is also a type of messenger, distributing load between processing cores.

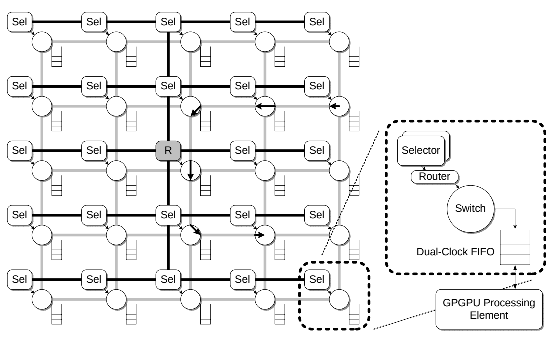

Fig. 1. Illustration of Hermes Selectors and Switches

Each Hermes node comprises three main modules: (1) a Dual-Clock FIFO, (2) a Switch/Router, and (3) two Selectors, as shown in Figure 1. The Selectors' job is to choose the source and destination nodes of the next job to be transferred. Two Selectors are needed, one to choose the node with the largest job count (i.e., the source of the next job transfer) and one to choose the node with the smallest job count (i.e., the destination). The Switch configures itself in such a way as to make a path between the source and destination nodes. The Dual-Clock FIFO is the job queue, where jobs are stored. As the name suggests, the Dual-Clock FIFO has two clock domains: one is for the Switch and the other is for the GPU subsystem. This characteristic allows Hermes to accommodate processing elements with different operating frequencies. If a node is chosen by the Selector to be a source or a destination, its Switch is configured to route information to/from the Dual-Clock FIFO.

## 3. Experimental Results

To evaluate the proposed load balancer, a simulator has been developed with SystemC. Since low-level details of GPU architectures are not publically available, we assume a design that is as close as possible to the high-level architectures attributed to commercially available GPUs from NVIDIA and ATI.

We compared three alternative approaches – a blocking centralized queue (BL), a non-blocking centralized queue (NB), and a distributed queue with job-stealing (DQ) – to ours (Hermes).

The result shown in Figure 2 demonstrates that by using the existing techniques (BL, NB, and DQ) the total execution time drops to some degree, but starts to remain constant or even increase with an increasing number of processing elements. The same trend was also observed in the experiments of [1]. This result confirms that only Hermes is scalable, regardless of the length of the execution time.

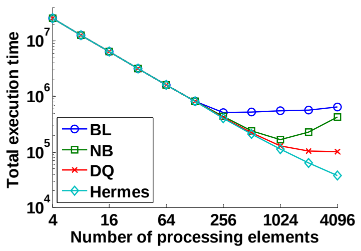

Fig. 2. The total execution time over increasing number of processing elements

## 4. Related Works

There has been prior work on hardware implementation of the scheduler in CPUs [3-4]. However, load distribution and balancing are not addressed.

Carbon [2] implements load balancing, as well as a scheduler, in hardware. Carbon employs centralized job queues (contained in the Global Task Unit). To hide latency between the queues and the cores, Carbon uses task pre-fetchers and small associated buffers close to the cores (called Local Task Units). However, as the cores scale to well over one hundred, contention at the Global Task Unit is expected to be excessive.

## 5. Conclusions

One of the key emerging challenges in GPGPU is to ensure that the GPU is scalable with the number of processing elements. This paper proposes a novel hardware-based dynamic load distributor and balancer, called Hermes, which is both scalable with as many as 1024 processing elements.

## References

[1] D. Cederman and P. Tsigas, "On dynamic load balancing on graphic processors," in *Proceedings of the 23rd ACM symposium on Graphics hardware*, 2008, pp.57-64
[2] S. Kumar, C. Hughes, and A. Nguyen, "Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors," in *Proceedings of the 34th annual International Symposium on Computer Architecture*, USA, 2007, pp. 162-173
[3] J. Agron *et al*., "Run-time services for hybrid CPU/FPGA systems on chip," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, Brazil, 2006, pp. 3-12
[4] P. Kuacharoen, M. A. Shalan and V. J. Mooney III, "A configurable hardware scheduler for real-time systems," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, USA, 2003
[5] M. Korch, and T. Rauber, "A comparison of task pools for dynamic load balancing of irregular algorithms," *Concurrency and Computation: Practice & Experience*, Volume 16, Issue 1, pp. 1-47, 2003