Affinity-Aware Work-Stealing for Integrated CPU-GPU Processors

Naila Farooqui
Georgia Institute of Technology
naila@cc.gatech.edu

Rajkishore Barik
Intel Labs
rajkishore.barik@intel.com

Brian T. Lewis
Intel Labs
brian.t.lewis@intel.com

Tatiana Shpeisman
Intel Labs
tatiana.shpeisman@intel.com

Karsten Schwan
Georgia Institute of Technology
schwan@cc.gatech.edu

Abstract

Recent integrated CPU-GPU processors like Intel's Broadwell and AMD's Kaveri support hardware CPU-GPU shared virtual memory, atomic operations, and memory coherency. This enables fine-grained CPU-GPU work-stealing, but architectural differences between the CPU and GPU hurt the performance of traditionally-implemented work-stealing on such processors. These architectural differences include different clock frequencies, atomic operation costs, and cache and shared memory latencies. This paper describes a preliminary implementation of our work-stealing scheduler, Libra, which includes techniques to deal with these architectural differences in integrated CPU-GPU processors. Libra's affinity-aware techniques achieve significant performance gains over classically-implemented work-stealing. We show preliminary results using a diverse set of nine regular and irregular workloads running on an Intel Broadwell Core-M processor. Libra currently achieves up to a 2× performance improvement over classical work-stealing, with a 20% average improvement.

1. Introduction

On multi-core CPUs, work-stealing [1, 3–5] efficiently load-balances CPU cores by enabling idle cores to steal work from other ones. A traditional work-stealing implementation assigns a work-queue to each CPU hardware thread, and distributes the workload, divided into chunks, between those queues. At runtime, each hardware thread executes work chunks from its own queue until that becomes empty, at which time it begins stealing work from other queues. This continues until all queues are empty and the computation is complete.

Today's widespread integrated CPU-GPU processors combine a CPU and a GPU with different power/performance characteristics to improve overall energy use and performance. However, effectively leveraging the compute capabilities of both the CPU and GPU in integrated processors has been a challenge [2, 6–8]. Nevertheless, recent hardware improvements have made effective, finer-grain computation possible using both the CPU and GPU. Processors such as Intel's Broadwell and Skylake and AMD's Kaveri and Carrizo offer hardware CPU-GPU shared virtual memory (SVM), memory coherency, and atomic operations. That hardware support makes it possible to implement fine-grain work-stealing on integrated CPU-GPU processors.

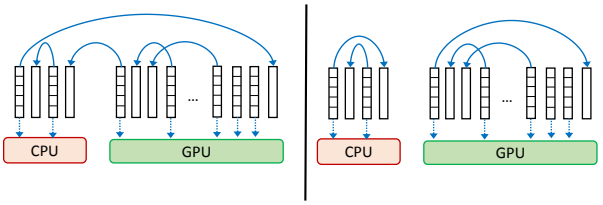

Figure 1: Classical work-stealing (left) and Libra's affinity-aware work-stealing that uses hierarchical stealing (right).

However, implementing CPU-GPU work-stealing efficiently is difficult: GPUs typically operate at lower clock frequencies than CPUs and have different core configurations and memory hierarchies, making their performance different than CPUs by an order of magnitude or more. In particular, because CPUs typically have lower-latency paths to local caches as well as lower-latency atomic instructions, a CPU is usually faster than a GPU at stealing work from another work queue *even if it is not the best device to execute that work*. As a result, while classically-implemented work-stealing is able to dynamically balance work across compute units, application performance may still suffer.

This paper describes our initial implementation of an affinity-aware CPU-GPU work-stealing scheduler, *Libra*, on an Intel Broadwell Core-M processor. Libra utilizes this processor's hardware shared virtual memory (SVM), memory coherency, and CPU-GPU atomics support. To improve work-stealing performance despite the architectural differences between the CPU and GPU, Libra uses two key techniques: 1) lightweight online profiling to initially distribute work across the CPU and GPU, and 2) *hierarchical work-stealing* to limit stealing by the device (typically the CPU) with lower performance executing the workload. Using a set of nine regular and irregular workloads, we demonstrate that Libra significantly improves performance over classical work-stealing: up to a 2× better with an average 20% improvement.

PPoPP '16 March 12-16, 2016, Barcelona, Spain

DOI: http://dx.doi.org/10.1145/2851141.2851194

Abbrev	Name	Input
MM	Matrix-Multiply	2048 by 2048
NB	N-Body	4096 bodies
SL	Skip-List	10M keys
BFS	Breadth-First Search	G3(1.6M V, 3M E)
SPMV	Sparse-Matrix Vector	Clueweb(100M V, 2B E)
SS	Substring-Finder	28657 items
BH	Barnes-Hut	1M bodies
BH-U	Barnes-Hut (unsorted)	1M bodies
FD	Face-Detect	image 3000x2171

Table 1: Benchmark characteristics

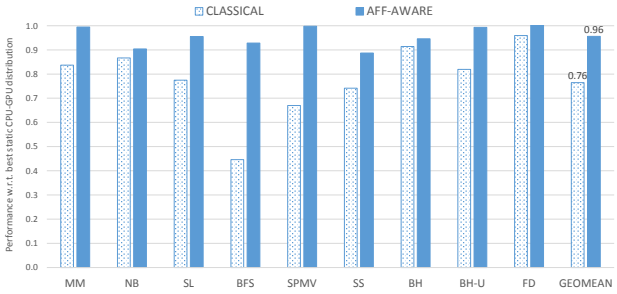

Figure 2: Performance relative to the static Oracle CPU-GPU work distribution, for classical work-stealing and affinity-aware work-stealing.

2. Our Approach

The goal of our Libra work-stealing runtime is to efficiently and dynamically balance data-parallel computation across the cores of CPU and GPU. To achieve this, our runtime: (1) maps high-level parallel computations to OpenCL work-groups; (2) binds the work-groups to physical cores; (3) assigns work-stealing deques to work-groups; and (4) finally, implements work-stealing among the work-groups.

Libra starts by launching a different number of work-groups on each device based on the device's hardware characteristics. For the CPU, we launch one work-group per hardware thread, and for the GPU, we launch one work-group per execution unit.

Classical work-stealing assumes that stealing has a uniform cost across all workers. In an integrated CPU-GPU processor, however, this assumption is typically incorrect. In addition, since the CPU and GPU have different execution characteristics, many applications exhibit a strong affinity to (execute better on) one device over the other. Since device affinity may also depend on input data, determining application affinity must be determined at runtime.

We use lightweight online profiling to determine device affinity at runtime for a given workload-input pair. This lets us optimize initial work placement, which reduces overall runtime steals. It especially minimizes stealing between devices, and so helps to reduce CPU-GPU contention caused by steal attempts.

In addition, we use *hierarchical stealing* to minimize stealing by the device (typically the CPU) with lower affinity. This also lowers cross-device stealing. In this technique, the worker threads on each device are restricted to stealing only from the same device as long as any work remains on one of its deques. Only when all deques on the same device are empty is a worker thread allowed to steal from the other device's deques. Figure 1 depicts how classical work-stealing and affinity-aware work-stealing differ when stealing.

3. Evaluation

We evaluated Libra on a laptop computer with 8GB of memory running 64-bit Windows 8. It includes a 1.2GHz Intel Broadwell (5th generation) Core-M 5Y71 processor with two CPU cores and an integrated Intel HD Graphics 5300 GPU with 24 execution units (EUs). The GPU has a maximum clock speed of 900 MHz and each of its EUs have seven 16-wide SIMD hardware threads. We enabled hyper-threading on the system.

Our evaluation used a diverse set of benchmarks spanning a spectrum of application domains and runtime characteristics, especially their execution regularity. An "irregular" workload has execution imbalance across different chunks of computation. The benchmarks are listed in Table 1.

We show the performance of both Libra and classic work-stealing compared to the baseline performance using a near-ideal, statically-determined Oracle work distribution. The Oracle static distribution is obtained via exhaustive search: we ran each benchmark with different fixed CPU-GPU work partitions, varying the percentage of work given to the CPU from 0% to 100% in steps of 10%, and selected the best-performing CPU-GPU work distribution.

In summary, our affinity-aware Libra work-stealing scheduler overcomes limitations of classically-implemented work-stealing for integrated CPU-GPU processors. It currently outperforms classical work-stealing by an average of 20% and is up to 2× better on some workloads. It also performs on par with the near-ideal Oracle static distribution. As a dynamic load-balancing scheduler, Libra can cope with execution-time irregularity caused by, e.g., differences in input data. We are continuing to improve our techniques and to evaluate Libra's use on other integrated processors.

References

[1] Intel thread building blocks. URL `www.threadbuildingblocks.org`.

[2] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23 (2):187–198, 2011.

[3] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, Sept. 1999. ISSN 0004-5411. doi: 10.1145/324133.324234.

[4] Y. Guo, J. Zhao, V. Cave, and V. Sarkar. Slaw: A scalable locality-aware adaptive work-stealing scheduler for multi-core systems. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 341–342, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-877-3. doi: 10.1145/1693453.1693504. URL `http://doi.acm.org/10.1145/1693453.1693504`.

[5] S. jai Min, C. Iancu, and K. Yelick. Hierarchical work stealing on many-core clusters. In *In Fifth Conference on Partitioned Global Address Space Programming Models*, 2011.

[6] R. Kaleem, R. Barik, T. Shpeisman, B. T. Lewis, C. Hu, and K. Pingali. Adaptive heterogeneous scheduling for integrated gpus. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, pages 151–162, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2809-8. doi: 10.1145/2628071.2628088. URL `http://doi.acm.org/10.1145/2628071.2628088`.

[7] J. Lee, M. Samadi, Y. Park, and S. Mahlke. Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, PACT, 2013.

[8] C.-K. Luk, S. Hong, and H. Kim. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 45–55, NY, USA, 2009. ACM. ISBN 978-1-60558-798-1. doi: 10.1145/1669112.1669121. URL `http://doi.acm.org/10.1145/1669112.1669121`.