Rodinia: A Benchmark Suite for Heterogeneous Computing

LAVA: Laboratory for Computer Architecture at Virginia
University of Virginia, Charlottesville, VA 22904
Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Sang-Ha Lee, Jeremy W. Sheaffer, Kevin Skadron
`http://lava.cs.virginia.edu`

This work is supported by a grant from NVIDIA Research and NSF grant nos. IIS-0612049 and CNS-0615277.

Motivations

With the microprocessor industry's shift to multicore architectures, research in parallel computing is essential to ensure future progress in mainstream computer systems. This in turn requires standard benchmark programs to compare platforms, identify performance bottlenecks, and evaluate potential solutions. Several current benchmark suites provide parallel programs, but only for conventional, general-purpose CPU architectures.

However, various accelerators (e.g. GPUs) are increasingly popular because they are becoming easier to program and offer dramatically better performance for many applications. These accelerators differ significantly from CPUs in architecture, middleware and programming model. Understanding these accelerators' architectural strengths and weaknesses is important for gaining insight into the most effective data structures and algorithms for each platform. Yet there is no benchmark suite we are aware of that provides a diverse set of applications for GPUs.

We present Rodinia benchmark suite, a set of applications and kernels we have developed for research to address these concerns. The application descriptions and codes of Rodinia are released online at http://lava.cs.virginia.edu/wiki/rodinia

Application Domains

Rodinia extends applications described in our previous analysis of GPU performance [1], and we have also done preliminary comparisons with FPGAs for several applications [2]. We are structuring the suite to span a range of parallelism and data-sharing characteristics according to the Berkeley's motifs. Rodinia currently includes nine applications in seven dwarves and nine application domains for both GPUs and multicore CPUs using CUDA and OpenMP.

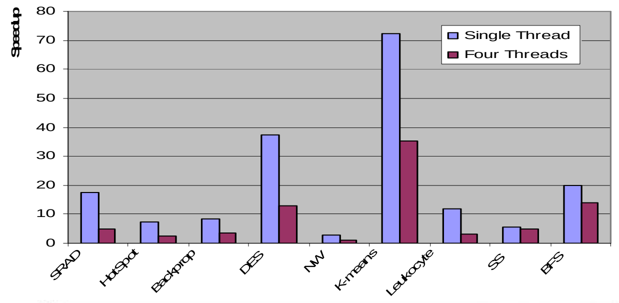
Table 1 Applications, kernels and domains (* denotes kernel)

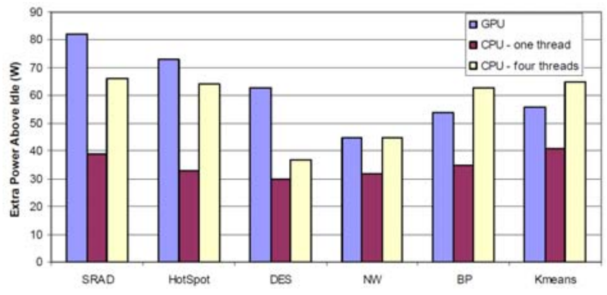
App/Kernel	Dwarves	Domains
Leukocyte Det./Track	Structured Grid	Medical Imaging
SRAD	Structured Grid	Physics Simulation
HotSpot*	Structured Grid	Image Processing
Back Propagation*	Unstructured Grid	Pattern Recognition
Needleman Wunsch	Dynamic Programming	Bioinformatics
K-means	Dense Linear Algebra	Data Mining
DES	Combinational Logic	En/Decryption
Breadth-First Search*	Graph Traversal	Graph Algorithms
Similarity Scores*	MapReduce	Web Mining

[1] S. Che, et al., "A Performance Study of General-Purpose Applications on Graphics Processors using CUDA," J. Parallel Distrib. Comput., Oct. 2008
[2] S. Che, et al., "Accelerating Compute Intensive Applications with GPUs and FPGAs," SASP 2008.

Measurement Results

The benchmarks have been evaluated on an NVIDIA GeForce GTX 280 GPU with 1.3 GHz shader clock and a multi-core Intel Xeon CPU with two 3.2 GHz hyperthreaded dual-core processors.





The Rodinia benchmarks exhibit a variety of behaviors, for example, speedups range from 2.9 to over 72.5 over single-thread CPU programs and from 1.0 to 35.4 over four-thread CPU programs, varying synchronization overheads (5ms-110ms), and varying power consumption overheads (40W-83W).

Architectural Challenges and Optimizations

• The GPU's lack of persistent state in the shared memory results in less efficient communication among producer and consumer kernels.

• GPUs do not easily allow runtime load balancing of work among threads within a kernel, and thread resources can be wasted as a result (e.g., Needleman-Wunsch).

• GPUs also offer no global barrier other than a new kernel call. Discrete GPUs also have high kernel-call and data-transfer costs.

• The GPU offers a very low ratio of on-chip storage to number of threads, but also offers specialized memory spaces that can mitigate these costs — the shared memory (e.g., SRAD, HotSpot and DES), constant (e.g. Leukocyte), and texture memories (e.g., Kmeans).

• We use a multi-iteration ghost-zone technique, trading off redundant computation to reduce expensive synchronization between thread blocks (e.g., HotSpot)

• We use lookup tables to avoid high SIMD divergence penalties when an application performs irregular branching in implementing bit-level permutations (e.g., in DES).

Future Work

We plan to extend our Rodinia work by including more applications from other dwarves (e.g., sorting and sparse matrix) and by comparing applications written in other languages (e.g., OpenCL and VHDL). To ensure the Rodinia benchmarks are well distributed within the workload space and suitable for various platforms, we will evaluate our benchmark programs using architectural-independent metrics such as instruction mix, branch behavior, register traffic characteristics, computation-to-communication ratio and so on.