# Experience Porting General-Purpose Applications to GPUs using CUDA

*LAVA: Laboratory for Computer Architecture at Virginia*
*University of Virginia, Charlottesville, VA 22904*
**Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, 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.

## Introduction

Graphics processors (GPUs) have become increasingly attractive for general purpose computation as they provide massive parallelism, high memory bandwidth, and general purpose instruction sets. They often contain hundreds of cores and are the best commercially-available example of "many core" processors today.

### Motivation

CUDA and GPUs allow study of massively parallel, shared memory programs on commodity hardware. CUDA also offers a novel programming model. Our group is studying scaling bottlenecks in manycore architectures, and this poster summarizes our experiences using CUDA for a variety of applications, as reported in [1]. Instead of evaluating a single application, we explore a set of applications with different parallelism and data-sharing characteristics.

### Rodinia Benchmark Suite

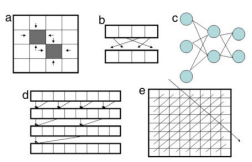
We are releasing our codes as the Rodinia benchmark suite for research in heterogeneous computing, including OpenMP and some FPGA implementations. Other codes and implementations will be added as they are completed. http://lava.cs.virginia.edu/wiki/rodinia

[1] S. Che, et al., A performance study of general-purpose applications on graphics processors using CUDA, *J. Parallel and Distributed Computing,* Elsevier, 2008.

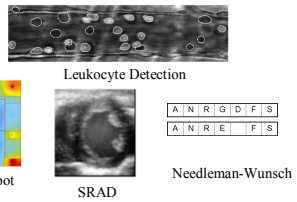
## Application Domains

We use Berkeley's "dwarf" taxonomy to select a representative range of application behaviors. Each dwarf represents a set of algorithms with similar computation and data movement patterns.

| | |
|---|---|
| **Structured Grid** | **– Leukocyte, SRAD and HotSpot** |
| **Unstructured Grid** | **– Back Propagation** |
| **Dynamic Programming** | **– Needleman-Wunsch** |
| **Dense Linear Algebra** | **– Kmeans** |
| **Combinational Logic** | **– DES** |



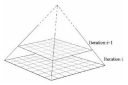

**Applications with various sharing patterns**




**Applications from different domains**

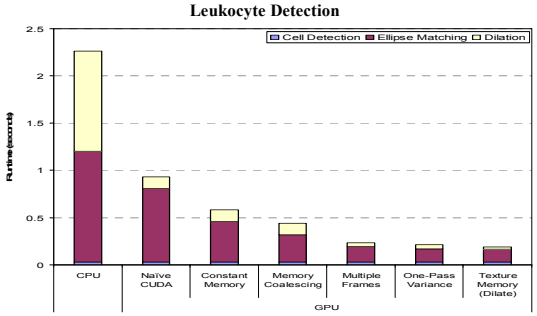
Communication patterns of different applications: (a) in SRAD and HotSpot, the value of each point depends on its neighboring points; (b) DES involves many bit-level permutations; (c) Back Propagation works on a layered neural network; (d) in SRAD and Back Propagation, parallel reductions can be performed using multiple threads; (e) the parallel Needleman–Wunsch algorithm processes the score matrix in diagonal strips

## Common Optimization Techniques



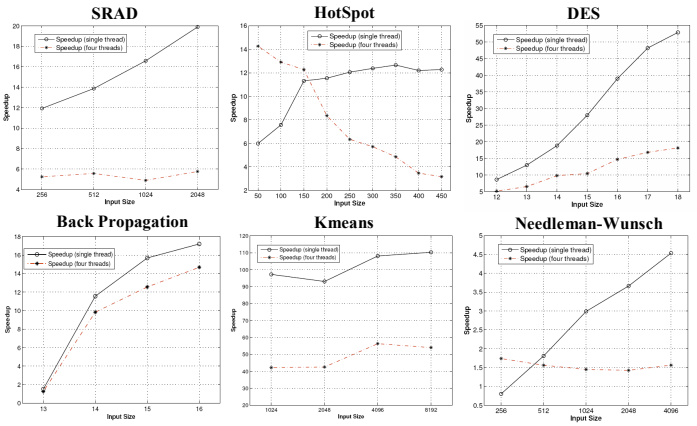
- Iterative solvers, such as our HotSpot thermal solver, benefit from a pyramidal data structure, trading off redundant computation to avoid the synchronization between thread blocks
- Lookup tables are useful for avoiding a large penalty due to branch divergence within SIMT groups (e.g., in the irregular data permutations of DES)
- Localizing data access patterns and inter-thread communication within thread blocks takes advantage of the SM's per-block shared memory (useful in most applications)
- Frequently accessed, read-only values shared across a warp should be placed in cached constant memory (e.g., in Leukocyte).
- Large, read-only data structures with temporal and spatial locality should be accessed as textures to exploit texture caching (e.g., in Kmeans)
- Data access patterns should be organized so that warps access contiguous blocks of memory (e.g., in Leukocyte).

**Leukocyte Detection**




For example, performance of Leukocyte Detection improves by about 30% with constant memory and Kmeans by about 70% with textures.

## Application Results




The GPU implementations are developed using CUDA and the multithreaded CPU versions using OpenMP. We compare application performance between an NVIDIA Geforce GTX 280 and an Intel Xeon CPU with two hyperthreaded dual-core processors (3.2 GHz, 2 MB L2 and 4GB main memory). The power result is obtained by subtracting the system power consumed when system is idling (213W) from the system power consumed during execution, and uses an early engineering sample of a GTX 260.

**Power Consumption**