How To Run Your CUDA Program Anywhere
– A Perspective from Virginia Tech’s GPU-Accelerated HokieSpeed Cluster

Wu Feng

Dept. of Computer Science
Dept. of Electrical & Computer Engg.
Virginia Bioinformatics Institute
Virginia Tech

Dept. of Cancer Biology and the
Translational Science Institute
School of Medicine
Wake Forest University

© W. Feng and M. Gardner, November 2011
Why High-Performance Computing (HPC)?

Competitive Risk From Not Having Access to HPC

- Only 3% of companies could exist and compete without HPC.
  - 200+ participating companies, including many Fortune 500 companies
    (Proctor & Gamble and biological and chemical companies)

Data from Council of Competitiveness. Sponsored Survey Conducted by IDC
Why GPU-Accelerated HPC?

The Second Coming of the “Beowulf Cluster” for HPC

• The First Coming of the “Beowulf Cluster” (Early 1990s)
  – Aggregated commodity PC technology to create what was dubbed a Beowulf cluster, delivering supercomputing to the masses.

• The Second Coming of the “Beowulf Cluster” (Early 2010s)
  – Leveraging commodity PC technology again, but …
    … this time in the form of commodity GPUs …
“Beowulf 1.0 → Beowulf 2.0”
Green Supercomputers (Nov. 2010)

<table>
<thead>
<tr>
<th>Green500 Rank</th>
<th>MFLOPS/W</th>
<th>Site*</th>
<th>Computer*</th>
<th>Total Power (KW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1684.20</td>
<td>IBM Thomas J. Watson Research Center</td>
<td>NNSA/SC Blue Gene/Q Prototype</td>
<td>38.80</td>
</tr>
<tr>
<td>2</td>
<td>1448.03</td>
<td>National Astronomical Observatory of Japan</td>
<td>GRAPE-DR accelerator Cluster, Infiniband</td>
<td>24.59</td>
</tr>
<tr>
<td>3</td>
<td>958.35</td>
<td>GSIC Center, Tokyo Institute of Technology</td>
<td>HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows</td>
<td>1243.80</td>
</tr>
<tr>
<td>4</td>
<td>933.06</td>
<td>NCSA</td>
<td>Hybrid Cluster Core i3 2.93GHz Dual Core, NVIDIA C2050, Infiniband</td>
<td>36.00</td>
</tr>
<tr>
<td>5</td>
<td>828.67</td>
<td>RIKEN Advanced Institute for Computational Science</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect</td>
<td>57.96</td>
</tr>
<tr>
<td>6</td>
<td>773.38</td>
<td>Universitaet Wuppertal</td>
<td>QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus</td>
<td>57.54</td>
</tr>
<tr>
<td>6</td>
<td>773.38</td>
<td>Universitaet Regensburg</td>
<td>QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus</td>
<td>57.54</td>
</tr>
<tr>
<td>6</td>
<td>773.38</td>
<td>Forschungszentrum Juelich (FZJ)</td>
<td>QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus</td>
<td>57.54</td>
</tr>
<tr>
<td>9</td>
<td>740.78</td>
<td>Universitaet Frankfurt</td>
<td>Supermicro Cluster, QC Opteron 2.1 GHz, ATI Radeon GPU, Infiniband</td>
<td>385.00</td>
</tr>
<tr>
<td>10</td>
<td>677.12</td>
<td>Georgia Institute of Technology</td>
<td>HP ProLiant SL390s G7 Xeon 6C X9660 2.8GHz, nVidia Fermi, Infiniband QDR</td>
<td>94.40</td>
</tr>
<tr>
<td>11</td>
<td>636.36</td>
<td>National Institute for Environmental Studies</td>
<td>GOSAT Research Computation Facility, nvidia</td>
<td>117.15</td>
</tr>
</tbody>
</table>
HokieSpeed
A GPU-Accelerated Supercomputer for the Masses

• Purpose
  – To catalyze new approaches for conducting research via the synergistic amalgamation of heterogeneous CPU-GPU hardware and software

• Profile
  – Total Nodes: 209, where each compute node consists of
    ▪ Motherboard: Supermicro 2026GT-TRF Dual Intel Xeon
    ▪ CPUs: Two 2.4-GHz Intel Xeon E5645 6-core (12 CPU cores per node)
    ▪ GPUs: Two NVIDIA Tesla Fermi GPUs (M2050/C2050)
HokieSpeed: Performance at a Glance

• An Instantiation of “The Second Coming of Beowulf Clusters”
• A Commodity GPU-Accelerated Supercomputer
  
  – **Speed**
    - Single-Precision Peak: 430.7 Tflops (Double-Precision Peak: 215.4 Tflops)
    - TOP500 Linpack: 120.4 Tflops
  
  – **Greenness**
    - CPU TDP: 85 W. GPU TDP: 225 W (M2050) and 238 W (C2050).
    - **Green500 Rating: 928.96 MFLOPS/W**
      - Green LINPACK: 117.3 Tflops
      - Total Power: 126.27 kW (including the network and disks)
HokieSpeed Team

- P. Balaji, CS @ ANL and U. Chicago
- S. Beardsley, Bio @ Marquette U.
- K. Bisset, CS & Social Sciences @ VT
- A. Butt, CS & Green Computing @ VT
- Y. Cao, CS & Social Sciences, VT
- N. Cellinese, Bio @ U. Florida
- I. Chen, Data Mining & Social Sciences @ VT
- T. D. Crawford, Chemistry @ VT
- Eric de Sturler, Mathematics @ VT
- W. Feng (PI), CS, Green Computing, Bio, and Data Mining @ VT
- X. Feng, CS & Bio @ Marquette U.
- M. Gardner, Office of IT @ VT
- R. Ge, CS & Green Computing @ Marquette U.
- K. Hilu, Bio @ VT
- D. Hong, Mechanical Engg. @ VT
- C. Hsu, CS & Green Computing @ ORNL
- S. King, Geosciences @ VT
- H. Lin, CS & Bio @ VT
- C. Lu, CS, Data Mining, & Social Sciences @ VT
- M. Marathe, Bio & Social Sciences @ VT
- P. McCormick, CS @ LANL
- D. Nikolopoulos, CS & Green Computing @ ICS-Forth
- A. Onufriev, Biochemistry & CS @ VT
- N. Ramakrishnan, Data Mining @ VT
- A. Sandu, Chemistry & CS @ VT
- J. Setubal, Bio @ VT
- C. Struble, Bio @ Marquette U.
- D. Tafti, Mechanical Engg. @ VT
- E. Tilevich, CS @ VT
- C. Weiss, Geosciences @ VT
- L. Zhang, CS & Bio @ VT
- P. Zhang, Bio @ VT
- Y. Zhou, Geosciences @ VT

© W. Feng and M. Gardner, November 2011
Forecast

• Motivation & Background
• Programming GPGPUs
• How To Run Your CUDA Program Anywhere
  – CU2CL: A CUDA-to-OpenCL Source-to-Source Translator
• Evaluation
  – Coverage, Translation Time, and Performance
• Future Work
• Summary
Programming GPGPUs

CUDA
- NVIDIA’s framework that popularized GPGPU programming

OpenCL
- An open standard (Khronos Group) that provides a vendor-neutral environment to program GPUs, CPUs, FPGAs, etc.
Challenges for Domain Scientists & Engineers

• **Novices:** Writing from Scratch → Learning Curve
  – OpenCL too low level an API compared to CUDA
    … easier to start with CUDA

• **Experts:** Leveraging CUDA Investment

• Porting from CUDA to OpenCL
  – Tedious and error-prone
Initialization Code: CUDA

None!
Initialization Code: OpenCL

1. clGetPlatformIDs(1, &clPlatform, NULL);
2. clGetDeviceIDs(clPlatform, CL_DEVICE_TYPE_GPU, 1, &clDevice, NULL);
3. clContext = clCreateContext(NULL, 1, &clDevice, NULL, NULL, &errcode);
4. clCommands = clCreateCommandQueue(clContext, clDevice, 0, &errcode);
5. kernelFile = fopen("srad_kernel.cl", "r");
6. fseek(kernelFile, 0, SEEK_END);
7. kernelLength = (size_t) ftell(kernelFile);
8. kernelSource = (char *) malloc(sizeof(char)*kernelLength);
9. rewind(kernelFile);
10. fread((void *) kernelSource, kernelLength, 1, kernelFile);
11. fclose(kernelFile);

12. clProgram = clCreateProgramWithSource(clContext, 1, (const char **) kernelSource, &kernelLength, &errcode);
13. free(kernelSource);
14. clBuildProgram(clProgram, 1, &clDevice, NULL, NULL, NULL);

15. clKernel_srad = clCreateKernel(clProgram, "srad_cuda_1", &errcode);
16. clKernel_srad2 = clCreateKernel(clProgram, "srad_cuda_2", &errcode);
Executing Device Code: CUDA

1. dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);

2. for (int i=0; i < matrix_dim-BLOCK_SIZE; i += BLOCK_SIZE) 

3. { 

4.     lud_diagonal<<<1, BLOCK_SIZE>>>(m, matrix_dim, i);

5.     lud_perimeter<<<(matrix_dim-i)/BLOCK_SIZE-1, BLOCK_SIZE*2>>>(m, matrix_dim, i);

6.     dim3 dimGrid((matrix_dim-i)/BLOCK_SIZE-1, (matrix_dim-i)/BLOCK_SIZE-1);

7.     lud_internal<<<dimGrid, dimBlock>>>(m, matrix_dim, i);

8. }

9. lud_diagonal<<<1,BLOCK_SIZE>>>(m, matrix_dim, i);
Executing Device Code: OpenCL

1. size_t localWorkSize[2];
2. size_t globalWorkSize[2];

3. for (int i=0; i < matrix_dim-BLOCK_SIZE; i += BLOCK_SIZE)
4. {
5.     clSetKernelArg(clKernel_diagonal, 0, sizeof(cl_mem), (void *) &d_m);
6.     clSetKernelArg(clKernel_diagonal, 1, sizeof(int), (void *) &matrix_dim);
7.     clSetKernelArg(clKernel_diagonal, 2, sizeof(int), (void *) &i);
8.     localWorkSize[0] = BLOCK_SIZE;
9.     globalWorkSize[0] = BLOCK_SIZE;
10.    clEnqueueNDRangeKernel(clCommands, clKernel_diagonal, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL);
     ... (14 more lines)
25. }
26. clSetKernelArg(clKernel_diagonal, 0, sizeof(cl_mem), (void *) &d_m);
27. clSetKernelArg(clKernel_diagonal, 1, sizeof(int), (void *) &matrix_dim);
28. clSetKernelArg(clKernel_diagonal, 2, sizeof(int), (void *) &i);
29. localWorkSize[0] = BLOCK_SIZE;
30. globalWorkSize[0] = BLOCK_SIZE;
31. clEnqueueNDRangeKernel(clCommands, clKernel_diagonal, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL);
Challenges for Domain Scientists & Engineers

• **Novices:** Writing from Scratch → Learning Curve
  – OpenCL too low level an API compared to CUDA
    … arguably easier to start with CUDA

• **Experts:** Leveraging CUDA Investment
Prevalence

First public release February 2007

First public release December 2008

Search Date: 2011-09-14
CUDA-Accelerated Applications

GOVERNMENT & DEFENSE
Ikena: Imagery Analysis and Video Forensics
Signal Processing Library: GPU VSIPL
IDL and MATLAB® Acceleration: GPULib
GIS: Manifold

MOLECULAR DYNAMICS, COMPUTATIONAL CHEMISTRY
OpenMM library for molecular dynamics on GPUs
GROMACS using OpenMM
NAMD molecular dynamics
VMD visualization of molecular dynamics
HOOMD molecular dynamics
Acellera: ACEMD bio-molecular dynamics package
BigDFT: DFT (Density functional theory) electronic structure
MDGPU
GPUGrid.net

LIFE SCIENCES, BIO-INFORMATICS
GPU HMMER
DNA Sequence alignment: MUMmerGPU
LISSOM: model of human neocortex using CUDA
Silicon Informatics: AutoDock

ELECTRODYNAMICS AND ELECTROMAGNETIC
Acelleware: FDTD Solver
Acelleware: EM Solutions
Remcom XStream FDTD

SPEAG Semcad X
CST Microwave Studio
Quantum electrodynamics library
GPMAD: Particle beam dynamics simulator

MEDICAL IMAGING, CT, MRI
RealityServer
GPULib:IDL acceleration
Acelleware: Imaging Solutions
Digisens: SnapCT tomographic reconstruction software
Techniscan: Whole Breast Ultrasound Imaging System

OIL & GAS
Acelleware: Kirchoff and Reverse Time Migration
SeismicCity: 3D seismic imaging for prestack depth migration
OpenGeoSolutions: Spectral decomposition and inversion
Mercury Computer systems: 3D data visualization
ffA: 3D Seismic processing software
Headwave: Prestack data processing

FINANCIAL COMPUTING AND OPTIONS PRICING
SciComp: derivatives pricing
Hanweck: options pricing
Exegy: Risk Analysis
Aqumin: 3D Visualization of market data
Level 3 Finance
OnEye (Australia): Accelerated Trading Solutions

Arbitragis Trading

MATLAB, LABVIEW, MATHEMATICA, R
CUDA Acceleration for MATLAB
Accelereyes: Jacket™ engine for MATLAB
GPULib: mathematical functions for IDL and MATLAB
Integrating Simulink with CUDA using S-functions
Enabling GPU Computing in the R Statistical Environment
Mathematica plug-in for CUDA
National Instruments LabView for NVIDIA GPUs

ELECTRONIC DESIGN AUTOMATION
Agilent EESof: ADS SPICE simulator
Synopsys: Sentarans TCAD
Gauda: Optical proximity correction (OPC)

WEATHER AND OCEAN MODELING
CUDA-accelerated WRF code

VIDEO, IMAGING, AND VISION APPLICATIONS
Axxon Intelent Enterprise Video Surveillance Software
Pflow CUDA Plugin for Autodesk 3ds Max
RUINS Shatter CUDA Plug-in for Maya
Bullet 3D Multi-Physics Library with CUDA Support
CUDA Voxel Rendering Engine
Furryball: Direct3D GPU Rendering Plugin for Maya
Examples of Available CUDA Source Code

- odeint: ODE solver
- OpenCurrent: PDE solver
- R+GPU: accelerate R
- Alenka: “SQL for CUDA”
- GPIUTMD: multi-particle dynamics
- rCUDA: remote invocation
- HOOMD-blue: particle dynamics
- Exact String Matching for GPU
- GMAC: asymmetric distributed memory
- TRNG: random number generation
- OpenNL: numeric library
- VMD: visual molecular dynamics
- CUDA memtest
- GPU-accelerated Ising model
- Image segmentation via Livewire
- OpenFOAM: accelerated CFD
- PFAC: string matching
- NBSimple: n-body code
- WaveTomography: wave propagation reconstruction
- CUDAFAST: cosmological lattice
- HPMC: volumetric iso-surface extraction
- OpenMM: molecular dynamics
- MUMmerGPU: DNA alignment
- SpMV4GPU: sparse-matrix multiplication toolkit
  and more…
Challenges for Domain Scientists & Engineers

• Novices: Writing from Scratch → Learning Curve
  – OpenCL too low level an API compared to CUDA
    … arguably easier to start with CUDA
• Experts: Leveraging CUDA Investment
• Our Solution
  – How To Run Your CUDA Program Anywhere via
    **CU2CL: A CUDA-to-OpenCL Source-to-Source Translator**
    (“cuticle”)
How To Run Your CUDA Program Anywhere

<table>
<thead>
<tr>
<th>CUDA Program</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CU2CL</strong></td>
</tr>
<tr>
<td>(”cuticle”)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OpenCL-supported CPUs, GPUs, FPGAs</th>
<th>NVIDIA GPUs</th>
</tr>
</thead>
</table>

Including NVIDIA GPUs!
Forecast

• Motivation & Background
• Programming GPGPUs
• How To Run Your CUDA Program Anywhere
  – CU2CL: A CUDA-to-OpenCL Source-to-Source Translator
• Evaluation
  – Coverage, Translation Time, and Performance
• Future Work
• Summary
Goals of CU2CL (‘‘cuticle’’)

• Automatically support many CUDA applications
• Maintainable OpenCL code for future development
• Seek adoption from CUDA and OpenCL communities
Ecosystem for “CUDA Anywhere”

- **Source Languages**
  - CUDA
  - OpenCL
  - Other

- **Device-Specific Driver**
  - NVIDIA GPU
  - AMD GPU
  - AMD APU
  - AMD CPU
  - Intel CPU

- **Translator Infrastructure**
  - PTX
  - CAL
  - ASM & CAL
  - ASM

- **PlaSorm-Specific Optimizations**
  - Language-Dependent Front Ends
  - Platform-Dependent De-Optimizations
  - Platform-Independent Optimizations
  - Platform-Specific Optimizations

© W. Feng and M. Gardner, November 2011
# CUDA & OpenCL API

<table>
<thead>
<tr>
<th>CUDA Module</th>
<th>OpenCL Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Contexts &amp; Command Queues</td>
</tr>
<tr>
<td>Device</td>
<td>Platforms &amp; Devices</td>
</tr>
<tr>
<td>Stream</td>
<td>Command Queues</td>
</tr>
<tr>
<td>Event</td>
<td>Events</td>
</tr>
<tr>
<td>Memory</td>
<td>Memory Objects</td>
</tr>
</tbody>
</table>
**CUDA & OpenCL Data**

<table>
<thead>
<tr>
<th>CUDA</th>
<th>OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector types (e.g. float4)</td>
<td>Host: cl_float4</td>
</tr>
<tr>
<td></td>
<td>Kernel: float4</td>
</tr>
<tr>
<td>dim3</td>
<td>size_t[3]</td>
</tr>
<tr>
<td>cudaStream_t</td>
<td>cl_command_queue</td>
</tr>
<tr>
<td>cudaEvent_t</td>
<td>cl_event</td>
</tr>
<tr>
<td>Device pointers (e.g. float* created through cudaMalloc)</td>
<td>cl_mem created through clCreateBuffer</td>
</tr>
<tr>
<td>cudaChannelFormat</td>
<td>cl_image_format</td>
</tr>
<tr>
<td>textureReference</td>
<td>cl_mem created through clCreateImage</td>
</tr>
<tr>
<td>cudaDeviceProp</td>
<td>No direct equivalent</td>
</tr>
</tbody>
</table>
### CUDA & OpenCL Synchronization Functions

#### Writing Kernels: Synchronization

<table>
<thead>
<tr>
<th>C for CUDA terminology</th>
<th>OpenCL terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>__syncthreads()</td>
<td>barrier()</td>
</tr>
<tr>
<td>__threadfence()</td>
<td>No direct equivalent.</td>
</tr>
<tr>
<td>__threadfence_block()</td>
<td>mem_fence(CLK_GLOBAL_MEM_FENCE</td>
</tr>
<tr>
<td>No direct equivalent.</td>
<td>read_mem_fence()</td>
</tr>
<tr>
<td>No direct equivalent.</td>
<td>write_mem_fence()</td>
</tr>
</tbody>
</table>
CUDA Language Overview

- Divides host (CPU) and device (GPU) code
- Both are extensions of C/C++
- Host-side APIs set up GPUs and launch kernels

**Runtime API**
- High-level API
- Implicitly performs most set up/tear down
- Assumes single device

**Driver API**
- Low-level API
- Requires explicit set up/tear down
- Allows multiple devices
OpenCL Language Overview

- Divides host (CPU) and device (GPU) code
- Generalized, device-agnostic API
- Support for task- and data-parallel kernels
- Kernels written in separate files
- No interplay between host and device code
  - No extern’d constant or shared memory
- No direct access to device pointers (CUDA 4.0)
  - Prevents device structs with device pointers
Translator Base to Build Upon

- Production-quality compiler
- Ease of extensibility

© W. Feng and M. Gardner, November 2011
The Clang Compiler Framework

- Useful libraries for C/C++ source-level tools
- Powerful AST representation
- Clang compiler built on top

© W. Feng and M. Gardner, November 2011
AST-Driven, String-Based Rewriting

• Characteristics
  – Does not modify the AST
  – Instead, edit text in source ranges

• Benefits
  – Useful for transformations with limited scope
  – Preserves formatting and comments
Architecture of CU2CL

Clang Framework

Clang Driver

Libraries Used

AST

AST

Traverse
Identify
Rewrite

Lex, Rewrite

OpenCL † Host Files
OpenCL † Kernel Files

CUDA § Source Files

* Abstract Syntax Tree
§ Compute Unified Device Architecture
† Open Computing Language

© W. Feng and M. Gardner, November 2011
Translation Procedure of CU2CL

• Traverse the AST
  – Clang’s AST library, walking nodes and children

• Identify structures of interest
  – Common patterns arise

• Rewrite original source range as necessary
  – Variable declarations: rewrite type
  – Expressions: recursively rewrite full expression
  – Host code: remove from kernel files
  – Device code: remove from host files
  – #includes: rewrite to point to new files
Rewriting `#includes`

```
Includer.cu
...
#include \ "CudaFile.cuh"
...
```

```
Clang Preprocessor
```

```
CU2CL
```

```
Includer-cl.cpp
#include \ "CudaFile-cl.h"
...
```

```
Includer-cl.cl
#include \ "CudaFile-cl.cl"
...
```
Rewriting #includes

```c
#include "CudaFile.cuh"
```

© W. Feng and M. Gardner, November 2011
Forecast

• Motivation & Background
• Programming GPGPUs
• How To Run Your CUDA Program Anywhere
  – CU2CL: A CUDA-to-OpenCL Source-to-Source Translator
• Evaluation
  – Coverage, Translation Time, and Performance
• Future Work
• Summary
Experimental Set-Up

- **CPU**
  - 2 x 2.0-GHz Intel Xeon E5405 quad-core
  - 4 GB of Ram

- **GPU**
  - NVIDIA GTX 280
  - 1 GB of graphics memory

- **Applications**
  - CUDA SDK
    - asyncAPI, bandwidthTest, BlackScholes, matrixMul, scalarProd, vectorAdd
  - Rodinia
    - Back Propagation, Breadth-First Search, Hotspot, Needleman-Wunsch, SRAD
# CUDA Coverage: CUDA SDK and Rodinia

<table>
<thead>
<tr>
<th>Source</th>
<th>Application</th>
<th>CUDA Lines</th>
<th>Changed</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA SDK</td>
<td>asyncAPI</td>
<td>136</td>
<td>4</td>
<td>97.06</td>
</tr>
<tr>
<td></td>
<td>bandwidthTest</td>
<td>891</td>
<td>9</td>
<td>98.99</td>
</tr>
<tr>
<td></td>
<td>BlackScholes</td>
<td>347</td>
<td>4</td>
<td>98.85</td>
</tr>
<tr>
<td></td>
<td>matrixMul</td>
<td>351</td>
<td>2</td>
<td>99.43</td>
</tr>
<tr>
<td></td>
<td>scalarProd</td>
<td>171</td>
<td>4</td>
<td>97.66</td>
</tr>
<tr>
<td></td>
<td>vectorAdd</td>
<td>147</td>
<td>0</td>
<td>100.00</td>
</tr>
<tr>
<td>Rodinia</td>
<td>Back Propagation</td>
<td>313</td>
<td>5</td>
<td>98.40</td>
</tr>
<tr>
<td></td>
<td>Breadth-First Search</td>
<td>306</td>
<td>8</td>
<td>97.39</td>
</tr>
<tr>
<td></td>
<td>Hotspot</td>
<td>328</td>
<td>7</td>
<td>97.87</td>
</tr>
<tr>
<td></td>
<td>Needleman-Wunsch</td>
<td>418</td>
<td>0</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>SRAD</td>
<td>541</td>
<td>0</td>
<td>100.00</td>
</tr>
</tbody>
</table>
CUDA Coverage: Molecular Modeling App

2,511 CUDA lines out of 6,727 total SLOC in GEM application

<table>
<thead>
<tr>
<th>Source</th>
<th>Application</th>
<th>CUDA Lines</th>
<th>Changed</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginia Tech</td>
<td>GEM</td>
<td>2,511</td>
<td>5</td>
<td>99.8</td>
</tr>
</tbody>
</table>

- **Fundamental Application in Computational Biology**
  - Simulate interactions between atoms & molecules for a period of time by approximations of known physics

- **Example Usage**
  - Understand mechanism behind the function of molecules
    - Catalytic activity, ligand binding, complex formation, charge transport
Model for Total Translation Time

Increase due to CU2CL: 0.87-2.2%
Model for CU2CL-Only Translation Time

![Graph showing the relationship between CUDA source lines (in thousands) and translation time (in milliseconds). The graph includes a linear regression line with the equation 6.467 * KLOC + 2.659.]

© W. Feng and M. Gardner, November 2011
### Status of OpenCL & the 13 Dwarfs

<table>
<thead>
<tr>
<th>Dwarf</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense linear algebra</td>
<td>LU Decomposition</td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Matrix Multiplication</td>
</tr>
<tr>
<td>Spectral methods</td>
<td>FFT</td>
</tr>
<tr>
<td>N-Body methods</td>
<td>GEM</td>
</tr>
<tr>
<td>Structured grids</td>
<td>SRAD</td>
</tr>
<tr>
<td>Unstructured grids</td>
<td>CFD solver</td>
</tr>
<tr>
<td>MapReduce</td>
<td></td>
</tr>
<tr>
<td>Combinational logic</td>
<td>CRC</td>
</tr>
<tr>
<td>Graph traversal</td>
<td>BFS, Bitonic sort</td>
</tr>
<tr>
<td>Dynamic programming</td>
<td>Needleman-Wunsch</td>
</tr>
<tr>
<td>Backtrack and Branch-and-Bound</td>
<td></td>
</tr>
<tr>
<td>Graphical models</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>Finite state machines</td>
<td>Temporal Data Mining</td>
</tr>
</tbody>
</table>

© W. Feng and M. Gardner, November 2011
## Translated Application Performance (sec)

<table>
<thead>
<tr>
<th>Application</th>
<th>CUDA</th>
<th>Automatic OpenCL</th>
<th>Manual OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectorAdd</td>
<td>0.0499</td>
<td>0.0516</td>
<td>0.0521</td>
</tr>
<tr>
<td>Hotspot</td>
<td>0.0177</td>
<td>0.0565</td>
<td>0.0561</td>
</tr>
<tr>
<td>Needleman-Wunsch</td>
<td>6.65</td>
<td>8.77</td>
<td>8.77</td>
</tr>
<tr>
<td>SRAD</td>
<td>1.25</td>
<td>1.55</td>
<td>1.54</td>
</tr>
</tbody>
</table>

- Automatically translated OpenCL codes yield similar execution times to manually translated OpenCL codes
- OpenCL performance lags CUDA (at least for OpenCL 1.0)
CU2CL with OpenCL and the 13 Dwarfs

<table>
<thead>
<tr>
<th>Dwarf</th>
<th>Implemented</th>
<th>AMD GPU Unoptimized</th>
<th>NVIDIA GPU Unoptimized</th>
<th>AMD CPU Unoptimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense linear algebra</td>
<td>LU Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Matrix Multiplication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral methods</td>
<td>FFT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-Body methods</td>
<td>GEM</td>
<td>GEM</td>
<td>GEM</td>
<td>GEM</td>
</tr>
<tr>
<td>Structured grids</td>
<td>SRAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured grids</td>
<td>CFD Solver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapReduce</td>
<td>StreamMR</td>
<td>StreamMR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combinational logic</td>
<td>CRC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph traversal</td>
<td>BFS, Bitonic Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic programming</td>
<td>Needleman-Wunsch</td>
<td></td>
<td>Smith-Waterman</td>
<td></td>
</tr>
<tr>
<td>Backtrack and Branch-and-Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical models</td>
<td>Hidden Markov Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finite state machines</td>
<td>Temporal Data Mining</td>
<td></td>
<td>TDM</td>
<td></td>
</tr>
</tbody>
</table>

© W. Feng and M. Gardner, November 2011
OpenCL Optimization Potential

<table>
<thead>
<tr>
<th></th>
<th>GTX280</th>
<th>Tesla C2050 (Fermi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup over hand-tuned SSE</td>
<td>328</td>
<td>317</td>
</tr>
<tr>
<td>CUDA</td>
<td>201</td>
<td>186</td>
</tr>
<tr>
<td>OpenCL</td>
<td>61.3%</td>
<td>58.7%</td>
</tr>
</tbody>
</table>

© W. Feng and M. Gardner, November 2011
### Status of OpenCL & the 13 Dwarfs

<table>
<thead>
<tr>
<th>Dwarf</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense linear algebra</td>
<td>LU Decomposition</td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Matrix Multiplication</td>
</tr>
<tr>
<td>Spectral methods</td>
<td>FFT</td>
</tr>
<tr>
<td>N-Body methods</td>
<td>GEM</td>
</tr>
<tr>
<td>Structured grids</td>
<td>SRAD</td>
</tr>
<tr>
<td>Unstructured grids</td>
<td>CFD solver</td>
</tr>
<tr>
<td>MapReduce</td>
<td></td>
</tr>
<tr>
<td>Combinational logic</td>
<td>CRC</td>
</tr>
<tr>
<td>Graph traversal</td>
<td>BFS, Bitonic sort</td>
</tr>
<tr>
<td>Dynamic programming</td>
<td>Needleman-Wunsch</td>
</tr>
<tr>
<td>Backtrack and Branch-and-Bound</td>
<td></td>
</tr>
<tr>
<td>Graphical models</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>Finite state machines</td>
<td>Temporal Data Mining</td>
</tr>
</tbody>
</table>

© W. Feng and M. Gardner, November 2011
Translator Ecosystem

Device-Specific Driver

Source Translator Infrastructure Languages

Language-Dependent Front Ends
Platform-Dependent De-optimizations
Platform-Independent Optimizations
Platform-Specific Optimizations

CUDA
OpenCL
Other

PTX
CAL
ASM & CAL
ASM
ASM

NVIDIA GPU
AMD GPU
AMD APU
AMD CPU
Intel CPU
Potential Due to Optimization

Platform awareness enhances performance portability

NVIDIA GTX280  AMD 5870

© W. Feng and M. Gardner, November 2011
The Bigger Picture

Design & Compile Time

- Computational & Communication Patterns: 13 Dwarfs

Source-to-Source Translation & Optimization Framework

Run Time

- Affinity Cost Models
- Task Scheduling System

Architecture-Aware Optimizations

- Performance & Power Models

OpenACC “pre-1.0”
CU2CL: Acknowledgments

• Collaborators
  – Gabriel Martinez, M.S. (now at Intel)
  – Mark Gardner, Ph.D.

• Infrastructure
  – Clang compiler and LLVM framework
Contributions

1. General approach for translating CUDA to OpenCL
2. Robust CUDA to OpenCL translator
   - Less than 2000 source lines of code
   - Extends open-source Clang compiler/framework
   - AST-driven, string-based source rewriting → maintainable OpenCL code
3. Already a useful tool
   - Eliminates most hand translation of CUDA 3.x constructs
   - Translated OpenCL performance = hand-translated

• Will be presented at ICPADS in December
• Already receiving many inquiries about release
CU2CL: Lessons Learned

1. Automatic-translated code and hand-translated code from CUDA to OpenCL yields the same performance
2. OpenCL performance is not as good as CUDA as implementations are not as mature
3. A translation ecosystem for performance portability is possible

Questions?