On the Three P’s of Heterogeneous Computing with Accelerators

Wu FENG
Dept. of Elec. & Comp. Engg., Dept. of Computer Science, Virginia Bioinformatics Institute
[Dept. of Cancer Biology and Translational Science Institute at Wake Forest University]
The Three P’s of Heterogeneous Computing?

- **Performance**
  - How fast does your code run?

- **Power**
  - How much power (or energy) does your code consume?

- **Programmability**
  - How easy was your code to program?

- **Price**
  - How much did the system cost to acquire?
  - How much does the system cost to operate and maintain?

- **Portability**
  - How many different systems can your code run on?
Japanese ‘Computnik’ Earth Simulator Shatters U.S. Supercomputer Hegemony

**Tokyo 20 April 2002** The Japanese Earth Simulator is online and producing results that alarm the USA, that considered itself as being leading in supercomputing technology. With over 35 Tflop/s, it five times outperforms the Asci White supercomputer that is leading the current TOP500 list. No doubt that position is for the Earth Simulator, not only for the next list, but probably even for
Importance of High-End Computing (HEC)

- **Trend**
  - Heterogeneous HEC dominates the top 10 of the

  "On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
  © W. Feng, 2011
## November 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>1243.60</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 VIlfx 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>7</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>8</td>
<td>773.38</td>
<td>Forschungszentrum Juellich (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>365.00</td>
</tr>
<tr>
<td>10</td>
<td>677.12</td>
<td>Georgia Institute of Technology</td>
<td>HP ProLiant SL390s G7 Xeon 6C X5660 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>
Moore’s Law
Moore’s Law for Power Density

Unsustainable in the Long Term

- Sun’s Surface
- Rocket Nozzle
- Nuclear Reactor
- Heat Plate
- Pentium®

Source: S. Borkar, Intel

"On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
CPU Core Counts ...

- Doubling every 18-24 months
  - 2006: 2 cores
    - Examples: AMD Athlon 64 X2, Intel Core Duo
  - 2010: 8-12 cores
    - Examples: AMD Magny Cours, Intel Nehalem EX

- Penetrating all “cluster-on-a-chip” markets ...
  - Desktops
  - Laptops: Most in this room are multicore
  - Tablets: Apple iPad 2, HP TX1000, Sony S2
  - Cell Phones: LG Optimus 2X, Motorola Droid X2

A world of ubiquitous parallelism ...

... how to extract performance ... and then scale out
Paying For Performance

• “The free lunch is over...” †
  – Programmers can no longer expect substantial increases in single-threaded performance.
  – The burden falls on developers to exploit parallel hardware for performance gains.

• How do we lower the cost of concurrency?

The Berkeley View†

• Traditional Approach
  – Applications that target existing hardware and programming models

• Berkeley Approach
  – Hardware design that keeps future applications in mind
  – Basis for future applications?

  13 dwarfs
    • 7 original by P. Colella
    • 6 additional by Asanovic et al.

Project Goal
An Ecosystem for Heterogeneous Computing

• Deliver personalized heterogeneous supercomputing to the masses
  – Heterogeneity of hardware devices for a “cluster on a chip” plus ...
  – Enabling software that tunes the parameters of the hardware devices with respect to performance, power, and programmability via a benchmark suite of computational dwarfs and apps
Project Outcome

An Ecosystem for Heterogeneous Computing

• A multi-dimensional understanding of how to optimize performance, power, programmability, or some combination thereof
  – Performance (under Resource Constraints)
    • # threads/block; # blocks/grid; configurable memory; mixed-mode arithmetic; and so on
  – Power
    • Device vs. system power
    • Instantaneous vs. average power consumption
  – Programmability
    • OpenCL vs. CUDA (NVIDIA) vs. Verilog/ImpulseC (Convey)

• What about portability?
An Ecosystem for Heterogeneous Parallel Computing

Science & Engineering Principles

- Math Algorithms: Solvers, Optimization, Model Reduction, Dynamic Multi-Precision Support
- Exascale Simulation Framework
- Verification, Validation, and Uncertainty Quantification

Design of Composite Structures

Software Ecosystem

Design & Compile Time
- Computational & Communication Patterns: 13 Dwarfs
- Source-to-Source Translation & Optimization Framework
- Architecture-Aware Optimizations

Run Time
- Affinity Cost Models
- Task Scheduling System
- Performance & Power Models

Heterogeneous Parallel Computing (HPC) Platform

"On the Three P's of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
Roadmap

- OpenCL and the 13 Dwarfs
- Source-to-Source Translation
- Architecture-Aware Optimizations
- Heterogeneous Task Scheduling

All 3 P’s

“Programmability”

Performance

All 3 P’s
Project Goal

An Ecosystem for Heterogeneous Computing

- Deliver personalized heterogeneous supercomputing to the masses
  - Heterogeneity of hardware devices for a “cluster on a chip” plus ...
  - Enabling software that tunes the parameters of the hardware devices with respect to performance, power, and programmability via a benchmark suite of computational dwarfs and apps
What Is “OpenCL and the 13 Dwarfs”?  

OpenCL: Open Computing Language  
– A framework for writing programs that execute ... across heterogeneous computing platforms, ... consisting of CPUs, GPUs, or other processors.

The 13 Dwarfs  
– Original 7  
  • Dense linear algebra (e.g. dense matrix multiply)  
  • Sparse linear algebra (e.g. sparse matrix solvers)  
  • Spectral methods (e.g. FFT)  
  • N-Body methods (e.g. gravity simulations)  
  • Structured grids (e.g. PDEs)  
  • Unstructured grids (e.g., irregular grids)  
  • MapReduce (e.g., data parallelism & combination)  

– 6 More Dwarfs  
  • Combinational logic  
  • Graph traversal  
  • Dynamic programming  
  • Backtrack & branch-and-bound  
  • Graphical models  
  • Finite state machines
## Status of OpenCL and the 13 Dwarfs

<table>
<thead>
<tr>
<th>Dwarf</th>
<th>Done</th>
<th>In progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense linear algebra</td>
<td>LU Decomposition</td>
<td></td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Matrix Multiplication</td>
<td></td>
</tr>
<tr>
<td>Spectral methods</td>
<td>FFT</td>
<td></td>
</tr>
<tr>
<td>N-Body methods</td>
<td>GEM</td>
<td>RRU (RoadRunner Universe)</td>
</tr>
<tr>
<td>Structured grids</td>
<td>SRAD</td>
<td></td>
</tr>
<tr>
<td>Unstructured grids</td>
<td>CFD Solver</td>
<td></td>
</tr>
<tr>
<td>MapReduce</td>
<td></td>
<td>PSICL-BLAST, StreamMR</td>
</tr>
<tr>
<td>Combinational logic</td>
<td>CRC</td>
<td></td>
</tr>
<tr>
<td>Graph traversal</td>
<td>BFS, Bitonic Sort</td>
<td></td>
</tr>
<tr>
<td>Dynamic programming</td>
<td>Needleman-Wunsch</td>
<td>N-Queens, Traveling Salesman</td>
</tr>
<tr>
<td>Backtrack and Branch-and-Bound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical models</td>
<td>Hidden Markov Model</td>
<td></td>
</tr>
<tr>
<td>Finite state machines</td>
<td>Temporal Data Mining</td>
<td></td>
</tr>
</tbody>
</table>
Selected Applications and their Dwarfs …

• FFT: Fast Fourier Transform
  – A *spectral method* that is used for a myriad of signal processing applications, e.g., video surveillance, etc.

• Electrostatic Surface Potential (ESP) of Molecules
  – An *n-body method* calculation to support molecular dynamics
  – Popular packages: AMBER, GROMACS, LAMPPS

• Smith-Waterman: Local Sequence Alignment
  – A *dynamic programming method*
    • The *full computation*, including matrix filling and storing, affine gap-penalty calculations, and *backtracing*.

• N-Queens
  – A *backtrack method* … meant to highlight benefits of FPGA

"On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
# Initial Performance: Actual & Estimated (Nov. 2010)

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>FPGA Convey</th>
<th>GPU</th>
<th>GPU</th>
<th>GPU</th>
<th>GPU</th>
<th>GPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESP (MPPS)</td>
<td>Serial</td>
<td>HC-1 ex 1 V5 LX760</td>
<td>HC-1 ex 1 V5 LX330</td>
<td>AMD Radeon HD 5450</td>
<td>AMD Radeon HD 5870</td>
<td>NVIDIA ION (Zotac)</td>
<td>NVIDIA 320M (Apple)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Serial</td>
<td>4.6</td>
<td>4800.0</td>
<td>2400.0</td>
<td>41.5</td>
<td>2119.0</td>
<td>141.4</td>
</tr>
<tr>
<td></td>
<td>FFT 1D-Float (GFLOPS)</td>
<td>3.3</td>
<td>3.3</td>
<td>240.3</td>
<td>38.4</td>
<td>-</td>
<td>379.2</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>FFT 2D-Float (GFLOPS)</td>
<td>1.6</td>
<td>1.6</td>
<td>144.0</td>
<td>26.5</td>
<td>-</td>
<td>111.6</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FFT 1D-Int (GOPS)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Smith-Waterman (MCUPS)</td>
<td>14.2</td>
<td>14.2</td>
<td>688,000</td>
<td>-</td>
<td>7.6</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

MPPS: Million Pairs Per Second  
GFLOPS: Giga Floating-Point Operations per Second  
MCUPS: Million Cell Updates Per Second  

Numbers in *italics* are architecture-optimized.  

Estimated (or from Convey) in gray  

**IMPORTANT!** See “Caveats” slide.
Caveats and Notes

- GPU Implementation of Smith-Waterman: A complete implementation with matrix filling, matrix storage, affine gap penalties, and backtrace, and thus, not directly comparable to other implementations... many of which ignore the backtrace.
- FPGA performance #s for ESP based on area & speed data of Xilinx FP operators in actual computation pipeline @ 300MHz. Actual implementation is in place but facing some issues.
- Floating-point FFT for GPU & FPGA: 1D-1024pt, 32-bit and 2D-512*512 pts, 32-bit. (FPGA @ 300MHz, GPU @ 800+MHz.)
- Integer FFT for Convey HC-1: 64pt, 16-bit @ 150MHz.
- Device costs are not considered. High-end FPGA: 50x-80x more $$$ than a high-end GPU. Data transfer costs are ignored for "estimated" FPGA performance #s.
- FPGA assumption for FFT: Overhead to do a transpose for 2D FFT is negligible.
- Projected performance with all four devices active for Convey would be ~4X for both 1D and 2D.
- FFT numbers in grey are estimates. FFT IP cores are configured in streaming/pipelined mode. In V6 all 8 memory ports are used, thus performance is memory-bound.
- FFT numbers in green are actual numbers from device. All 8 memory ports are used. Streaming FFTs replaced by Radix-2 FFTs containing fewer DSP48E slices to have a balanced implementation across all memory ports. Performance is limited by the number of DSP48E slices (Total of 192 on V5LX330T). Numbers are low due to reduced real read/write memory BW obtained (~15.5 GB/s) mainly due to memory stalls on interfaces (expected 20GB/s per FPGA).
Performance Update: 1-D FFT (Jun. 2011)

Note: Host-to-device transfer overhead ignored in all cases.
Performance: N-Queens (Backtrack)

<table>
<thead>
<tr>
<th>N</th>
<th>Execution time in seconds</th>
<th>AMD Radeon HD 5870 GPU **</th>
<th>NVidia Fermi GPU **</th>
<th>Convey HC - 1 FPGA *</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.04</td>
<td>0.13</td>
<td>0.000017</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.04</td>
<td>0.14</td>
<td>0.00004</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.04</td>
<td>0.16</td>
<td>0.00015</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.05</td>
<td>0.16</td>
<td>0.00071</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.05</td>
<td>0.18</td>
<td>0.0036</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.06</td>
<td>0.19</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.04</td>
<td>0.22</td>
<td>0.119</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.05</td>
<td>0.24</td>
<td>0.746</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.04</td>
<td>0.25</td>
<td>5.02</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.09</td>
<td>0.29</td>
<td>35.6</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.62</td>
<td>1.16</td>
<td>266.41</td>
<td></td>
</tr>
</tbody>
</table>

* Estimated execution time based on implementation on a single Virtex 5 FPGA
Power: Thermal Design Power (TDP)

- Single Xeon: 130 Watts
- GTX 280: 260 Watts
- Double Xeon: 236 Watts
- ION: 13 Watts
- HD 5870: 188 Watts
- HD 5450: 19 Watts

"On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
Total System Power: Idle vs. At Load (w/ FFT)

<table>
<thead>
<tr>
<th>System</th>
<th>Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core 2 Duo OpenCL (45 nm)</td>
<td>95, 141</td>
</tr>
<tr>
<td>Convey HC-1 4 X FPGA V5 LX330 (65nm)</td>
<td>271, 291</td>
</tr>
<tr>
<td>AMD Radeon HD 5870 (40nm)</td>
<td>188, 260</td>
</tr>
<tr>
<td>NVIDIA GeForce GTX 280 (65nm)</td>
<td>236, 350</td>
</tr>
<tr>
<td>NVIDIA Ion 2 Mac mini (40nm)</td>
<td>12, 32</td>
</tr>
<tr>
<td>AMD Zacate E-350 APU</td>
<td>12, 17</td>
</tr>
</tbody>
</table>

What’s this?
The Future is Fusion

Experimental Set-Up: Machines and Workload

<table>
<thead>
<tr>
<th>Platform</th>
<th>AMD Zacate APU</th>
<th>AMD Radeon HD 5870</th>
<th>AMD Radeon HD 5450</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream Processors</td>
<td>80</td>
<td>1600</td>
<td>80</td>
</tr>
<tr>
<td>Compute Units</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Memory Bus Type</td>
<td>NA</td>
<td>GDDR5</td>
<td>DDR3</td>
</tr>
<tr>
<td>Device Memory</td>
<td>192 MB</td>
<td>1024 MB</td>
<td>512 MB</td>
</tr>
<tr>
<td>Local Memory</td>
<td>32 KB</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>Max. Workgroup Size</td>
<td>256 Threads</td>
<td>256 Threads</td>
<td>128 Threads</td>
</tr>
<tr>
<td>Core Clock Frequency</td>
<td>492 MHz</td>
<td>850 MHz</td>
<td>675 MHz</td>
</tr>
<tr>
<td>Peak FLOPS</td>
<td>80 GFlops/s</td>
<td>2720 GFlops/s</td>
<td>104 GFlops/s</td>
</tr>
<tr>
<td>Host: Processor</td>
<td>AMD Engg. Sample @1.6 GHz</td>
<td>Intel Xeon E5405 @2.0 GHz</td>
<td>Intel Celeron 430 @1.8 GHz</td>
</tr>
<tr>
<td>System Memory</td>
<td>2 GB (NA)</td>
<td>2 GB DDR2</td>
<td>2 GB DDR2</td>
</tr>
<tr>
<td>Kernel</td>
<td>Ubuntu 2.6.35.22</td>
<td>Ubuntu 2.6.28.19</td>
<td>Ubuntu 2.6.32.24</td>
</tr>
</tbody>
</table>

- **OpenCL and the 13 Dwarfs**
  - Sparse Linear Algebra: SpMV
  - N-body: Molecular Modeling
  - Spectral: FFT
  - Dense Linear Algebra: Scan and Reduce (SHOC @ ORNL)

Amdahl’s Law for Different Parallel Devices

- Single Threaded
- Ideal Amdahl’s Law (4 cores)
- Achieved (Multicore)
- Achieved (Discrete GPU)
- Achieved (Fused GPU)

Time (ms)

Serial Time, Parallel Time, Overhead

"On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
Performance: Molecular Modeling (N-Body)

- APU reduces data transfer times for all problem sizes.
- The efficacy of the APU increases as the problem size increases, but ...
- The kernel executes fastest on discrete AMD 5870 due to more and faster GPU cores. The fused AMD Zacate APU is next fastest.
Performance: Reduction (Dense Linear Algebra)

- The execution time profiles of the fused AMD APU and the discrete AMD 5870 are complementary.
  - Fused AMD APU die with only 80 GPU cores significantly improves data transfer time.
  - Discrete AMD 5870 die with 1600 faster GPU cores significantly improves kernel execution time.
System Power

- **AMD Fusion APU**
  - At idle: 12 watts
  - At load: 17 watts (Spectral Method: FFT)
  - At load: 20 watts (N-body: Molecular Modeling)

- **AMD Radeon HD 5870**
  Machine w/ 2-GHz Intel Xeon E5405
  - At idle: 188 watts
  - At load: 260 watts
## Total System Power: Idle vs. At Load (w/ FFT)

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>FPGA</th>
<th>GPU</th>
<th>AMD fused</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core 2 Duo OpenCL (45 nm)</td>
<td>95</td>
<td>141</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Convey HC-1 4 X FPGA V5 LX330 (65nm)</td>
<td>12</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD Radeon HD 5870 (40nm)</td>
<td>271</td>
<td>291</td>
<td>260</td>
<td>17</td>
</tr>
<tr>
<td>NVIDIA GeForce GTX 280 (65nm)</td>
<td>188</td>
<td>236</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>NVIDIA Ion 2 Mac mini (40nm)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD Zacate E-350 APU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"On the Three P's of Heterogeneous Computing," PPAC @ IEEE Cluster © W. Feng, 2011
## Status of OpenCL and 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>
Roadmap

- OpenCL and the 13 Dwarfs
- Source-to-Source Translation
- Architecture-Aware Optimizations
- Heterogeneous Task Scheduling

Shipping the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
Creating an Army of Dwarfs via “CU2CL” †

• **CU2CL:** **CUDA-to-OpenCL** Source-to-Source Translator
  – Implemented as a Clang plugin, allowing us to leverage its production-quality compiler framework and target LLVM bytecode.
  – Covers the primary CUDA constructs found in CUDA C and the CUDA run-time API.
  – Successfully translates CUDA applications from the CUDA SDK and Rodinia benchmark suite.
  – Performs as well as codes manually ported from CUDA to OpenCL.

• **Others:** **OpenCL-to-CUDA** and **OpenMP-to-OpenCL**

Why CU2CL?

- Much larger # of apps implemented in CUDA than in OpenCL
  - Idea
    - Leverage scientists’ investment in CUDA to drive OpenCL adoption
  - Issues (from the perspective of domain scientists)
    - Writing from Scratch: Learning Curve
      (OpenCL is too low-level an API compared to CUDA. CUDA also low level.)
    - Porting from CUDA: Tedious and Error-Prone
Initialization Code: CUDA

None!
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_srad1 = clCreateKernel(clProgram, "srad_cuda_1", &errcode);
16. clKernel_srad2 = clCreateKernel(clProgram, "srad_cuda_2", &errcode);
Why CU2CL?

• Much larger # of apps implemented in CUDA than in OpenCL
  – Idea
    • Leverage scientists’ investment in CUDA to drive OpenCL adoption
  – Issues (from the perspective of domain scientists)
    • Writing from Scratch: Learning Curve
      (OpenCL is too low-level an API compared to CUDA. CUDA also low level.)
    • Porting from CUDA: Tedium and Error-Prone

• Significant demand from major stakeholders

Why Not CU2CL?

▪ Just start with OpenCL?!
▪ CU2CL only does source-to-source translation at present
  • No architecture-aware optimization
  • CUDA and OpenCL version compatibility

At odds ...
Goals of CU2CL

- Automatically translate (or support) CUDA applications
- Generate maintainable OpenCL code for future development
Overview of CU2CL Translation

Experimental Set-Up

- **CPU**
  - 2 x 2.0-GHz Intel Xeon E5405 quad-core CPUs

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

- **Applications**
  - CUDA SDK: `asyncAPI, bandwidthTest, BlackScholes, matrixMul, scalarProd, vectorAdd`
  - Rodinia: `Back Propagation, Breadth-First Search, HotSpot, Needleman-Wunsch, SRAD`
## 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>

- Less optimized kernels account for longer runtimes in OpenCL
Roadmap

- OpenCL and the 13 Dwarfs
- Source-to-Source Translation
- Architecture-Aware Optimizations
- Heterogeneous Task Scheduling

All 3 P’s

“Programmability” Performance

All 3 P’s
Computational Units Not Created Equal

- “AMD CPU != Intel CPU” and “AMD GPU != NVIDIA GPU”
Summary: Architecture-Aware Optimization

![Graph showing speedup over hand-tuned SSE](chart.png)

- Basic speedup:
  - NVIDIA GTX280: 163
  - AMD 5870: 88

- Architecture aware speedup:
  - NVIDIA GTX280: 371
  - AMD 5870: 328

"On the Three P's of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011
Architecture-Aware Optimization
(N-body Code for Molecular Modeling)

• Optimization techniques on AMD GPUs
  – Removing conditions → kernel splitting
  – Local staging
  – Using vector types
  – Using image memory

• Speedup over basic OpenCL GPU implementation
  – Isolated optimizations
  – Combined optimizations
Roadmap

- OpenCL and the 13 Dwarfs  
  All 3 P’s
- Source-to-Source Translation  
  “Programmability”  
  Performance
- Architecture-Aware Optimizations
- Heterogeneous Task Scheduling  
  All 3 P’s

"On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster  
© W. Feng, 2011
What is Heterogeneous Task Scheduling?

• Automatically spreading tasks across heterogeneous compute resources
  – CPUs
  – GPUs
  – APUs

• Specify tasks at a higher level (currently OpenMP extensions)

• Run them across available resources automatically
Why Heterogeneous Task Scheduling?

• OpenCL is portable, running on
  – CPUs, CellBE, GPUs, APUs, and Intel MIC (future)
• Heterogeneous resource usage cannot be predicted at design or compile time
• “Architecture-Aware Optimizations” show that different systems cannot be optimized the same way.

Goal

• A run-time system that intelligently uses what is available resource-wise and optimize for performance portability
  – Each user should not have to implement this for themselves!
Experimental Testbed

- 12-core AMD Opteron 6174 CPU
- 1 NVIDIA Tesla C2050 GPU
- Linux 2.6.27.19
- CCE compiler with OpenMP accelerator extensions
KMeans with Heterogeneous Task Scheduling

Scheduler

- GPU
- Static
- Dynamic
- Split
- Quick

Speedup over 12-core CPU

© W. Feng, 2011

"On the Three P's of Heterogeneous Computing," PPAC @ IEEE Cluster
Roadmap

- OpenCL and the 13 Dwarfs
- Source-to-Source Translation
- Architecture-Aware Optimizations
- Heterogeneous Task Scheduling
  - Stay tuned for more on this soon ...

© W. Feng, 2011
Recent Publications on Heterogeneous Computing

People

• Research Staff
  – Mark K. Gardner, Ph.D.
  – Heshan Lin, Ph.D.

• Ph.D. Students
  – Ashwin Aji
  – Tom Scogland
  – Balaji Subramaniam
  – Shucai Xiao (graduating)
  – Jing Zhang

• M.S. Students
  – Mayank Daga (now @ AMD)
  – Gabriel Martinez (now @ Intel)

• B.S./M.S. Students
  – Carlo del Mundo
  – Tyler Kahn
  – Kenneth Lee
  – Paul Sathre
Wu Feng, wfeng@vt.edu, 540-231-1192

SyNeRG
http://synergy.cs.vt.edu/

CHREC
NSF Center for High-Performance Reconfigurable Computing
http://www.chrec.org/

mpiBLAST
http://www.mpiblast.org/

ENERGYGUIDE
SUPERCOMPUTING in SMALL SPACES
http://sss.cs.vt.edu/

"Accelerators ‘R Us"
http://accel.cs.vt.edu/

http://myvice.cs.vt.edu/

"On the Three P’s of Heterogeneous Computing," PPAC @ IEEE Cluster
© W. Feng, 2011