On the Complexity of Robust Source-to-Source Translation from CUDA to OpenCL

Paul D. Sathre
CS Graduate Seminar
April 26, 2013
Parallel and Heterogeneous Computing is *Everywhere*

Supercomputers to cell phones

Mac OS X (since Snow Leopard)
- GPU-accelerated

Adobe Photoshop CS6
- GPU-accelerated
GPU Performance

Speedups Using GPU vs CPU

146X: Interactive visualization of volumetric white matter connectivity
36X: Ionoc placement for molecular dynamics simulation on GPU
18X: Transcoding HD video stream to H.264 for portable video
17X: Simulation in Matlab using .mex file CUDA function
100X: Astrophysics N-body simulation

149X: Financial simulation of LIBOR model with swaptions
47X: GLAME@lab: M-script API for linear Algebra operations on GPU
20X: Ultrasound medical imaging for cancer diagnostics
24X: Highly optimized object oriented molecular dynamics
30X: Cmatch exact string matching - find similar proteins & gene sequences

Source: nvidia.com
GPU Computing

Major player in heterogeneous computing
Special-purpose languages for programming general purpose computations

- NVIDIA CUDA
- OpenCL
- Brook+, OpenACC, etc...

Specify host (CPU) code which controls device (GPU) code

OpenCL Devices

CUDA Devices

NVIDIA GPUs
AMD GPUs
Intel MIC
AMD Fused CPU/GPUs
Intel CPUs
and more...
Functional Portability
Overarching Vision

CUDA Source Code → Manual Translation (Days, weeks, years) → OpenCL Source Code

Automatic Translation (seconds) → CU2CL

CUDA Source Code → Automatic Translation (seconds) → OpenCL Source Code

Image Source: xkcd.com
The Importance of Openness

OpenCL – Open Standard
- Write once, run anywhere – for parallel programs
- Source code lives longer than hardware

Clang/LLVM – open source compiler efforts
- “Standing on the shoulders of giants”
- Robust CUDA to OpenCL translation in ~3400 lines of code
CU2CL Translator Prototype

Compilation Process

Preprocessor  
 Lexer  
 Parser  
 Semantic Analyzer  
 Code Generator  

Source Code  
 Preprocessed Code  
 Tokenized Code  
 Parse Tree  
 Intermediate Representation  
 Binary  

Clang  
 LLVM  

CU2CL Framework

Clang Driver

Clang Framework

Libraries Used

CUDA Source Files

Clang Framework

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

AST

Lex, Rewrite

Traverse  
 Identify  
 Rewrite  

OpenCL Kernel Files

OpenCL Kernel Files

Martinez, Gardener, and Feng, "CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-Core Architectures," ICPADS 2011
CU2CL Roadmap & Contributions

CU2CL Alpha (2011)
Well-designed scaffold

CU2CL Beta (2013)
Improved robustness, CUDA coverage, and reliability
Analysis and profiling of difficult-to-translate CUDA structures

CU2CL w/ Functional Portability
Expand CUDA coverage
- Shared, const, texture memory
- Driver API
- OpenGL
Handling unmapped CUDA structs/behaviors
- Warp sync

Automatic de-optimization
Device-agnostic optimization
Device-specific optimization

CU2CL w/ Performance Portability

Sathre, Gardner, Feng: "Lost in Translation: Challenges in Automating CUDA-to-OpenCL Translation". ICPP Workshops 2012

Gardner, Feng, Sathre, Martinez: "Characterizing the Challenges and Evaluating the Efficacy of a CUDA-to-OpenCL Translator". ParCo Special Issue 2013, to appear

CU2CL Beta Binary available to NSF CHREC Funding members since December.
Translation Is Easy ...

... when there is NO ambiguity in the translation between languages

High-level language $\rightarrow$ low-level representation, e.g., C $\rightarrow$ LLVM

\[ x \times y + z \rightarrow \]
\[ \%\text{tmp} = \text{mul} \ i32 \ \%x, \ \%y \]
\[ \%\text{tmp2} = \text{add} \ i32 \ \%\text{tmp}, \ \%z \]

Between languages, e.g., CUDA $\rightarrow$ OpenCL

```
__\text{powf}(x[\text{threadIdx}.x], \ y[\text{threadIdx}.y]) \rightarrow
\text{native}_\text{pow}(x[\text{get}\_\text{local}\_\text{id}(0)], \ y[\text{get}\_\text{local}\_\text{id}(1)])
```

String-based Recursive Rewriting

CUDA

```c
__powf(x[threadIdx.x], y[threadIdx.y])
```

OpenCL

```c
native_pow(x[get_local_id(0)], y[get_local_id(1)])
```

```
get_local_id(0)
```

```
get_local_id(1)
```
Translation isn’t easy

... when there IS ambiguity (or lack of one-to-one mapping) in the translation between languages

Idiomatic Expressions
- “Putting all your eggs in one basket” → ??
- CUDA `threadfence()` → ??

Dialects
- Latin American Spanish vs. Castilian Spanish → English
- CUDA Runtime API vs. CUDA Driver API → OpenCL
Complex Semantic Conversions

1. Literal Parameters to Kernels
   – CUDA pass-by-value invocations vs. OpenCL pass-by-reference

<table>
<thead>
<tr>
<th>CUDA Kernel Launch</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel &lt;&lt;&lt;grid, block &gt;&gt;&gt;(foo1, foo2 * 2.0f, 256);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Naive OpenCL Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>clSetKernelArg(__cu2cl_Kernel_kernel, 0, sizeof(float), &amp;foo1);</td>
</tr>
<tr>
<td>clSetKernelArg(__cu2cl_Kernel_kernel, 1, sizeof(float), &amp;foo2 * 2.0f);</td>
</tr>
<tr>
<td>clSetKernelArg(__cu2cl_Kernel_kernel, 2, sizeof(int), &amp;256);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correct OpenCL Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>clSetKernelArg(__cu2cl_Kernel_kernel, 0, sizeof(float), &amp;foo1);</td>
</tr>
<tr>
<td>float __cu2cl_Kernel_kernel_arg_1 = foo2 * 2.0f;</td>
</tr>
<tr>
<td>clSetKernelArg(__cu2cl_Kernel_kernel, 1, sizeof(float), &amp;__cu2cl_Kernel_kernel_arg_1);</td>
</tr>
<tr>
<td>int __cu2cl_Kernel_kernel_arg_2 = 256;</td>
</tr>
<tr>
<td>clSetKernelArg(__cu2cl_Kernel_kernel, 2, sizeof(int), &amp;__cu2cl_Kernel_kernel_arg_2);</td>
</tr>
</tbody>
</table>
Complex Semantic Conversions

2. Device Identification
   - CUDA uses `int`, OpenCL uses opaque `cl_device`
   - To change devices in CUDA, use `cudaSetDevice(int id)`
   - To change devices in OpenCL, use...

```c
// scan all devices
// save old platform, device, context, queue, program, & kernels
myDevice = allDevices[id]
ClGetDeviceInfo(...);  // get new device's platform
myContext = clCreateContext(...);
myQueue = clCreateCommandQueue(...);

// load program source
clBuildProgram(...);
myKernel = clCreateKernel(...);
```

- Implement our own handler to emulate and encapsulate

```c
/* CU2CL Warning -- CU2CL Identified cudaSetDevice usage*/
__cu2cl_SetDevice(devID);
```
Sample Population

79 CUDA SDK Samples

17 Rodinia Samples

GEM – Molecular Modeling
Fen Zi – Molecular Dynamics
IZ PS – Neural Network

100k+ SLOC in total

Image Sources:
http://www.ksuiuc.edu/Research/gpu/images/stmv.png
http://gclcis.udel.edu/projects/fenzi/movies/walp16.png
http://www.deviantart.com/download/177904574/the_neural_network2_by_rajasegar-d2xx41q.jpg
# Translation Challenges

## Profiled Challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>CUDA SDK Frequency (%)</th>
<th>Rodinia Frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device Identifiers</strong></td>
<td>54.4</td>
<td>29.4</td>
</tr>
<tr>
<td><strong>Literal Parameters</strong></td>
<td>19.0</td>
<td>23.5</td>
</tr>
<tr>
<td><strong>Separate Compilation</strong></td>
<td>54.4</td>
<td>29.4</td>
</tr>
<tr>
<td><strong>CUDA Libraries</strong></td>
<td>10.1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Kernel Templates</strong></td>
<td>21.5</td>
<td>0</td>
</tr>
<tr>
<td><strong>Texture Memory</strong></td>
<td>27.8</td>
<td>23.5</td>
</tr>
<tr>
<td><strong>Graphics Interoperability</strong></td>
<td>24.1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Constant Memory</strong></td>
<td>17.7</td>
<td>29.4</td>
</tr>
<tr>
<td><strong>Shared Memory</strong></td>
<td>46.8</td>
<td>70.6</td>
</tr>
</tbody>
</table>

## Other Identified Challenges

- Kernel Function Pointer Invocations
- Preprocessor Effects
- Warp-level Synchronization
- Device Intrinsic Functions
- Device Buffer `cl_mem` Type
- Propagation
- `#defined` Function Definitions
- Device Buffers as Struct Members
- Arrays of Device Buffers
- Implicitly-Defined Kernel Functions
- Device-side Classes, Constructors, & Destructors
- Struct Alignment Attributes

Sathre, Gardner, Feng: "Lost in Translation: Challenges in Automating CUDA-to-OpenCL Translation". ICPP Workshops 2012: 89-96

Gardner, Feng, Sathre, Martinez: "Characterizing the Challenges and Evaluating the Efficacy of a CUDA-to-OpenCL Translator". ParCo Special Issue 2013, to appear
## Translator Coverage

<table>
<thead>
<tr>
<th>Application</th>
<th>CUDA Lines</th>
<th>OpenCL Lines Changed</th>
<th>Percent Automatically Translated</th>
</tr>
</thead>
<tbody>
<tr>
<td>asyncAPI</td>
<td>135</td>
<td>5</td>
<td>96.3</td>
</tr>
<tr>
<td>bandwidthTest</td>
<td>891</td>
<td>5</td>
<td>98.9</td>
</tr>
<tr>
<td>BlackScholes</td>
<td>347</td>
<td>14</td>
<td>96.0</td>
</tr>
<tr>
<td>FastWalshTransform</td>
<td>327</td>
<td>30</td>
<td>90.8</td>
</tr>
<tr>
<td>matrixMul</td>
<td>351</td>
<td>9</td>
<td>97.4</td>
</tr>
<tr>
<td>scalarProd</td>
<td>251</td>
<td>18</td>
<td>92.8</td>
</tr>
<tr>
<td>vectorAdd</td>
<td>147</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Back Propagation</td>
<td>313</td>
<td>24</td>
<td>92.3</td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td>306</td>
<td>35</td>
<td>88.6</td>
</tr>
<tr>
<td>Gaussian</td>
<td>390</td>
<td>26</td>
<td>93.3</td>
</tr>
<tr>
<td>Hotspot</td>
<td>328</td>
<td>2</td>
<td>99.4</td>
</tr>
<tr>
<td>Needleman-Wunsch</td>
<td>430</td>
<td>3</td>
<td>99.3</td>
</tr>
<tr>
<td>Fen Zi</td>
<td>17768</td>
<td>1786</td>
<td>89.9</td>
</tr>
<tr>
<td>GEM</td>
<td>524</td>
<td>15</td>
<td>97.1</td>
</tr>
<tr>
<td>IZ PS</td>
<td>8402</td>
<td>166</td>
<td>98.0</td>
</tr>
</tbody>
</table>
Translated Application Performance

![Bar chart showing translated application performance with CUDA and OpenCL runtime comparisons for different benchmarks. The chart indicates that lower values are better, with benchmarks like SDK Samples and Rodinia Samples highlighted.](image-url)
CU2CL Reliability

Before Upgrades
- CUDA SDK Samples: 20.3% Complete, 11.4% Partial, 68.3% Failed
- Rodinia Samples: 52.9% Complete, 11.8% Partial, 35.3% Failed

After Upgrades
- CUDA SDK Samples: 20.3% Complete, 12.7% Partial, 21.5% Failed, 15.2% Clang 3.2, 24.1% Template handling
- Rodinia Samples: 52.9% Complete, 5.9% Partial, 23.5% Failed, 5.9% OpenGL #defined, 5.9% Separately declared and defined function handling, 5.9% Kernel pointer invocation handling

Legend:
- Failed
- Partial
- Complete
- Clang 3.2
- main() method handling
- Template handling
- OpenGL #defined function handling
- Separately declared and defined function handling
- Kernel pointer invocation handling
CU2CL Roadmap & Future Work

CU2CL Alpha (2011)

Well-designed scaffold

CU2CL Beta (2013)

Improved Robustness, CUDA Coverage, and Reliability

Analysis and profiling of difficult-to-translate CUDA structures

CU2CL w/ Functional Portability

Expand CUDA coverage
- Shared, const, texture memory
- Driver API
- OpenGL

Handling unmapped CUDA structs/behaviors
- Warp sync

CU2CL w/ Performance Portability

Automatic de-optimization

Device-agnostic optimization

Device-specific optimization
Conclusions

Robust automatic source translation from CUDA to OpenCL is (mostly) achievable
- Not straightforward, but moving towards full functional portability
- Some important (current) limitations
  - Libraries, Device C++, NVIDIA-specific functions/ hardware behaviors
  - Maturation of OpenCL ecosystem may help

Once translated, application performance is retained
- Work to utilize new devices is reduced
- Software is able to reach a broader audience

What used to take weeks/months/years to do by hand, now takes seconds!
Acknowledgements

Advisor: Dr. Wu-chun Feng

Collaborators: Dr. Mark Gardner, Gabriel Martinez

This work was supported in part by NSF I/UCRC IIP-0804155 via the NSF Center for High-Performance Reconfigurable Computing (CHREC).
Questions?

CUDA Source Code

Manual Translation (Days, weeks, years)

Automatic Translation (seconds)

CU2CL

OpenCL Source Code

Stay tuned for the online binary release of CU2CL ... coming May 2013.