CU2CL
Automated Source-to-Source Translation from CUDA to OpenCL for General Purpose Accelerated Computing
Synergy/SEEC Fall Symposium 2015
Paul Sathre
Parallel and Heterogeneous Computing is Everywhere

- Supercomputers to cell phones
- Mac OS X (since Snow Leopard) – GPU-accelerated
- Adobe Photoshop CS6 – GPU-accelerated

Heterogeneous Computing

System

CPU

Accelerator (GPU)

Offload data, computation

Work

Copy results back

Work
Functional Portability

- NVIDIA CUDA
- OpenCL
- Various hardware components
Translation Is Easy ...

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

• High-level language → low-level representation, e.g., C → LLVM
  
  \[
x * y + z \rightarrow \\
  \%
  \text{tmp} = \text{mul} \ i32 \ %x, \ %y \\
  \%
  \text{tmp2} = \text{add} \ i32 \ %\text{tmp}, \ %z
  \]

• Between languages, e.g., CUDA → OpenCL

  \_
  \_
  \text{powf}(x[\text{threadIdx.x}], \ y[\text{threadIdx.y}]) \rightarrow \\
  \text{native\_pow}(x[\text{get\_local\_id}(0)], \ y[\text{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
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

Translation Pipeline

- Traverse
- Identify
- Organize
- Rewrite

CU2CL

libClang

AST

Clang Components Utilized

- AST
- Replacement
- Lex
- Rewrite

CUDA Source Files

OpenCL Host Files

OpenCL Kernel Files

CU2CL Utility Files

Paul Sathre – SEEC Fall Symposium 2015.08.21
String-based Recursive Rewriting

- CUDA
  \[ \text{\texttt{\_\_powf(x[threadIdx.x], y[threadIdx.y])}} \]

- OpenCL
  \[ \text{\texttt{native_pow(x[get_local_id(0)], y[get_local_id(1)])}}} \]

<sourcefile>:<byteoffset>:<origlength>:"native_pow(x[get_local_id(0)], y[get_local_id(1)])"
## Translator Coverage

<table>
<thead>
<tr>
<th>Application</th>
<th>CUDA Lines</th>
<th>OpenCL Lines Changed</th>
<th>Percent Automatically Translated</th>
<th>Dwarf</th>
</tr>
</thead>
<tbody>
<tr>
<td>asyncAPI</td>
<td>135</td>
<td>5</td>
<td>96.3</td>
<td>-</td>
</tr>
<tr>
<td>bandwidthTest</td>
<td>891</td>
<td>5</td>
<td>98.9</td>
<td>-</td>
</tr>
<tr>
<td>BlackScholes</td>
<td>347</td>
<td>14</td>
<td>96.0</td>
<td>Dense Linear Algebra</td>
</tr>
<tr>
<td>Fast Walsh Transform</td>
<td>327</td>
<td>30</td>
<td>90.8</td>
<td>Spectral Methods</td>
</tr>
<tr>
<td>matrixMul</td>
<td>351</td>
<td>9</td>
<td>97.4</td>
<td>Dense Linear Algebra</td>
</tr>
<tr>
<td>scalarProd</td>
<td>251</td>
<td>18</td>
<td>92.8</td>
<td>Sparse/Dense Linear Algebra</td>
</tr>
<tr>
<td>vectorAdd</td>
<td>147</td>
<td>0</td>
<td>100</td>
<td>Sparse/Dense Linear Algebra</td>
</tr>
<tr>
<td>Back Propagation</td>
<td>313</td>
<td>24</td>
<td>92.3</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td>306</td>
<td>35</td>
<td>88.6</td>
<td>Graph Traversal</td>
</tr>
<tr>
<td>Gaussian</td>
<td>390</td>
<td>26</td>
<td>93.3</td>
<td>Dense Linear Algebra</td>
</tr>
<tr>
<td>Hotspot</td>
<td>328</td>
<td>2</td>
<td>99.4</td>
<td>Structured Grids</td>
</tr>
<tr>
<td>Needleman-Wunsch</td>
<td>430</td>
<td>3</td>
<td>99.3</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>Fen Zi</td>
<td>17768</td>
<td>1786</td>
<td>89.9</td>
<td>-</td>
</tr>
<tr>
<td>GEM</td>
<td>524</td>
<td>15</td>
<td>97.1</td>
<td>N-body Methods</td>
</tr>
<tr>
<td>IZ PS</td>
<td>8402</td>
<td>166</td>
<td>98.0</td>
<td>-</td>
</tr>
</tbody>
</table>
Translated Application Performance (Runtime Overhead)

Lower is Better

Test Systems:
- AMD Phenom II X6 + Nvidia GTX 480
- Intel i5 2400 + Nvidia K20C
- AMD Opteron 6272 + AMD Radeon 7970
Translated Application Performance: GEM

Lower is better!

Molecular Structure (smallest to largest)

- Mb.Helix.bondi
- 1uwo-A
- 1qks-A
- nucleosome
- 2eul
- capsid

Kernel Runtime (ms)

- NVIDIA K20C (CUDA)
- NVIDIA K20C (OpenCL)
- 7970 (OpenCL)
Future Efforts

• Modularize the frontend/backend
  - Separate CUDA structure identification from OpenCL string generation, so that potentially other languages can be "plugged in"

• De-C++ kernel code
  - De-templating
  - De-classing

• Altera/Non-standard CL mode
  - Generate offline compile scripts (could be useful for HSAIL)
  - Modify whatever boilerplate is required.

• Expanded CUDA support
  - Textures
  - Error handling
  - Dynamic Shared Memory
  - other API components
Related Work

• **Swan**: wrapper API around either CUDA OR OpenCL host API with a Basic CUDA-to-OpenCL Kernel Translator

• **MCUDA**: CUDA $\rightarrow$ OpenMP translator

• **Ocelot / Caracal**: PTX $\rightarrow$ LLVM IR / AMD CAL IR

• **CUDAtoOpenCL**: MS Project at UIUC: CUDA-to-OpenCL translator (AFAIK, never released, and now defunct)
Contributions

• Automatically translate upwards of 80–90% of most apps!
  - perfect translation in a few cases
• Maintainable code, retained formatting and comments
  - Translation in place maintains structure
• Collaboration and PR with AMD via translation of open source Fluidsv3 smooth particle hydrodynamics Visualization/Simulation
• Binary and source freely available for non-commercial use
• CHREC funders have had new 0.7.0b version since December
  - Unified translation of multi-file codebases
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 (i.e. SYCL)

• 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

• Advisors: Dr. Wu-chun Feng, Dr. Mark Gardner

• Alumni: Gabriel Martinez, Paul Sathre

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

Linux build and tool scripts available: http://chrec.cs.vt.edu/cu2cl
Source release available: https://github.com/vtsynergy/CU2CL

xkcd.com
Appendix
Globalized Translations

• Reimplemented plugin as *Clang Tool*
  - Can run consecutive translator instances on all source files making up a binary within a single invocation

• Intelligent cache and merge of rewritten code via *Replacement* objects
  
  \[
  \text{sourcefile:<byteoffset>:<origlength>:<replacementtext>}
  \]

  \[
  \text{fooKern.cu:1337:38:“native_pow(x[get_local_id(0)], y[get_local_id(1)])”}
  \]

• Deferred OpenCL boilerplate generation
  - partial initialization and deconstruction code generated per file
  - global initializers/deconstructors generated after last source file processed
Complex Semantic Conversions

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);
```
CU2CL Reliability

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

After Upgrades
- CUDA SDK Samples: 20.3% Failed, 12.7% Partial, 21.5% Clang 3.2 main() method handling, 15.2% Template handling, 24.1% #defined function handling, 2.5% Separately declared and defined function handling, 13% Kernel pointer invocation handling
- Rodinia Samples: 52.9% Failed, 5.9% Partial, 23.5% Clang 3.2 main() method handling, 5.9% Template handling, 5.9% #defined function handling, 5.9% Separately declared and defined function handling, 5.9% Kernel pointer invocation handling
AMD collaboration

Fluids v.3: Open Source Windows+CUDA fluid simulation, translated to OpenCL w/ CU2CL and run on AMD W9100s, demo at ISC’14 in Leipzig