One OpenCL to Rule Them All? *

Romain Dolbeau (CAPS),
François Bodin (Irisa),
Guillaume Colin de Verdière (CEA)
7 September 2013

*This work has been partially funded by the H4H-PerfCloud ITEA2 project http://www.h4h-itea2.org

Introduction

• OpenCL is available on a large set of computing platforms
  – Parallel model fits GPU and CPU constraints
  – Aims at avoiding machine specific development

• Code development performance-wise issues
  – Assessing portability of a code
  – Deciding development strategy and technology
    • e.g. auto-tuning, self-adapting code
Target Platforms for this Study

- **AMD7970**
  - 947 GFlops DP peak
  - 264 GB/s theoretical memory bandwidth (TMB)
  - 32 “compute units”, each with 4 “SIMDs”, having 16 scalar FPU
- **Nvidia K20C**
  - 1170 GFlops DP peak
  - 208 GB/s TMB
  - 13 “SMX” for a total of 2496 “CUDA cores”
- **Intel Xeon Phi SE10P**
  - 1070 GFlops DP peak
  - 352 GB/s TMB
  - 61 conventional superscalar in-order x86 64 cores, each capable of running 4 threads via HyperThreading

OpenCL Portability

- **Syntax**
  - High thanks to standard definition
- **Functional**
  - Not 100% due to explicit resources uses
  - e.g. local memory, threads block size, …
- **Performance**
  - ???
Which Development Strategy?

1. Optimize for each target independently

2. Search of a tradeoff performing well on all targets

3. Generate codes that can adapt to the target / runtime context

Development Strategy 1

- Produce one code for each target
  - Pros: target fitting, highest performance, predictability
  - Cons: Maintenance cost, application deployment
**Development Strategy 2**

- Offline tuning only
  - Pros: One code to deploy, maintenance cost, predictability
  - Cons: Performance averaging

**Development Strategy 3**

- Offline and online tuning strategy
  - Pros: Strategy 2 pros + performance, input set adaptability
  - Cons: Advanced technology, runtime overhead, debugging
Loss of Performance

• Definition
  – Efficiency Loss = \( \frac{\text{ExecTime} - \text{BestExecTime}}{\text{BestExecTime}} \)
  – 0% indicates that the variant reaches the best performance

• Pro
  – Measures efficiency of an approach compare to the best known code for a given target

• Con
  – Finding the best execution time is complicated and expensive

Experimental Application

• Hydro
  – Mini-application built from the RAMSES used to study large-scale structures and galaxy formation
  – Classical algorithms found in applications codes for Tier-0 systems

• Main properties for this study
  – Extensively studied on many targets (i.e. we know the best performance)
  – Non trivial with 22 OCL kernels (1.1)
  – Not communication bound
  – Code mainly sensitive to work-groups (threads grid) configuration
Code Generation Process

- Using CAPS OpenACC compiler
  - Simplifies tuning → Workgroup configurations
  - Hide implementation details for each targets
    - Simpler scan of the optimization space

Code Generation Process Assessment

- We compared the performance of
  - the generated OpenCL codes
  - to the native OpenCL version that was previously developed for Hydro

- The OpenCL generated code is
  - As fast as the hand-written version on the Phi
  - Slightly faster on the K20C
  - CUDA code instead of OpenCL for the K20C does not improve performance on this code

- Two work-groups settings
  - Same work-group configuration for all kernels
  - Kernel specific work-group sizes
Kernel Specific Work-group Sizes

- **AMD7970**
  - did not show much improvement (1.53%)

- **SE10P**
  - did not show much improvement (1.41%)

- **K20C**
  - showed an improvement of 8.39%
  - when optimizing for the K20C (or the SE10P) the code will no longer run on the AMD7970 as the work-group size used are too large

Efficiency Loss Results

<table>
<thead>
<tr>
<th>WG</th>
<th>AMD</th>
<th>Nvidia</th>
<th>Intel</th>
<th>Avg.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x4</td>
<td>6.5%</td>
<td>12%</td>
<td>23.5%</td>
<td>14%</td>
<td>23.5%</td>
</tr>
<tr>
<td>16x8</td>
<td>5%</td>
<td>5.5%</td>
<td>34.5%</td>
<td>15%</td>
<td>34.5%</td>
</tr>
<tr>
<td>16x16</td>
<td>7.5%</td>
<td>2.5%</td>
<td>38.5%</td>
<td>16%</td>
<td>38.5%</td>
</tr>
<tr>
<td>32x2</td>
<td>4.5%</td>
<td>22%</td>
<td>8%</td>
<td>11.5%</td>
<td>22%</td>
</tr>
<tr>
<td>32x4</td>
<td>3%</td>
<td>12.5%</td>
<td>11%</td>
<td>9%</td>
<td><strong>12.5%</strong></td>
</tr>
<tr>
<td>32x8</td>
<td>0%</td>
<td>6%</td>
<td>18%</td>
<td>8%</td>
<td>18%</td>
</tr>
<tr>
<td>64x1</td>
<td>9.5%</td>
<td>34%</td>
<td>3.5%</td>
<td>15.5%</td>
<td>33.5%</td>
</tr>
<tr>
<td>64x2</td>
<td>9%</td>
<td>20.5%</td>
<td>4.5%</td>
<td>11.5%</td>
<td>20.5%</td>
</tr>
<tr>
<td>64x4</td>
<td>4.5%</td>
<td>13%</td>
<td>6%</td>
<td>8%</td>
<td>13%</td>
</tr>
<tr>
<td>128x1</td>
<td>13%</td>
<td>27.5%</td>
<td>1%</td>
<td>13.5%</td>
<td>27.5%</td>
</tr>
<tr>
<td>128x2</td>
<td>7.5%</td>
<td>20.5%</td>
<td>2%</td>
<td>10%</td>
<td>20.5%</td>
</tr>
<tr>
<td>256x1</td>
<td>9.5%</td>
<td>28.5%</td>
<td>0%</td>
<td>12.5%</td>
<td>28.5%</td>
</tr>
<tr>
<td>16x32</td>
<td>N/A</td>
<td>0%</td>
<td>43.5%</td>
<td>21.5%</td>
<td><strong>43.5%</strong></td>
</tr>
</tbody>
</table>
### Data Highlights

<table>
<thead>
<tr>
<th></th>
<th>AMD</th>
<th>Nvidia</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>256x1</td>
<td>9.5%</td>
<td>28.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16x32</td>
<td>N/A</td>
<td>0%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**code non functional on AMD**

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32x4</td>
<td>3%</td>
<td>12.5%</td>
<td>11%</td>
</tr>
</tbody>
</table>

**Best configuration**

<table>
<thead>
<tr>
<th></th>
<th>16x32</th>
<th></th>
<th>43.5%</th>
<th>21.5%</th>
<th>43.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x8</td>
<td>0%</td>
<td>6%</td>
<td>10%</td>
<td>6%</td>
<td>10%</td>
</tr>
</tbody>
</table>

**Best for Nvidia worst for Intel**

|          | 256x1 |          | 0%       | 12.5%    | 28.5%    |

### Performance correlation between Nvidia and Intel Devices

- **Lower the better**

![Graph showing performance comparison between Nvidia and Intel devices](image-url)
Conclusion

• OpenCL a very interesting alternative for programming accelerator
• Code tuning remains an issue
• Performance tradeoff has to be explicitly researched
  – Performance portability can be achieved to an extent
  – No target can be used as a performance Oracle
• Auto-tuning techniques necessary to explore the optimization space
• Runtime techniques need to be very efficient to be worthwhile