Evaluating the performance of HPC-style SYCL applications

Tom Deakin
and Simon McIntosh-Smith

uob-hpc.github.io
Introduction

▪ SYCL was first released in 2014.
▪ Recent development of different implementations providing support for devices used in the HPC space.

▪ Platforms:
  – Intel Xeon Skylake and Iris Pro GPUs
  – NVIDIA RTX 2080 Ti GPU
  – AMD Radeon VII GPU

▪ Try out three different compilers:
  – Codeplay’s ComputeCpp
  – Intel’s oneAPI DPC++
  – Heidelberg University’s hipSYCL
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Architecture</th>
<th>Device Type</th>
<th>Mem. BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon Gold 6126 (12-core)</td>
<td>Skylake</td>
<td>HPC CPU (1 socket)</td>
<td>119.21</td>
</tr>
<tr>
<td>Intel NUC i7-6770HQ with Iris Pro 580 Graphics</td>
<td>Skylake/Gen9</td>
<td>CPU + Integrated GPU</td>
<td>34.1</td>
</tr>
<tr>
<td>AMD Radeon VII</td>
<td>Vega 20</td>
<td>Discrete GPU</td>
<td>1024</td>
</tr>
<tr>
<td>NVIDIA RTX 2080 Ti</td>
<td>Turing</td>
<td>Discrete GPU</td>
<td>616</td>
</tr>
</tbody>
</table>
Applications

- Three applications:
  - BabelStream
    - Copy kernel: $c[i] = a[i]$;
    - Triad kernel: $a[i] = b[i] + \text{scalar} \times c[i]$;
    - Dot kernel: $\text{sum} += a[i] \times b[i]$;
  - Heat
    - Simple explicit finite difference solve.
    - 5-point stencil.
  - CloverLeaf
    - 2D structured grid Lagrangian-Eulerian hydrodynamics code.

- All are main memory bandwidth bound, like many other HPC applications today.
BabelStream: Triad

- Results are shown as percentage of theoretical peak bandwidth, so higher is better.
- SYCL shows little overhead over direct implementations in the underlying models, particularly on the GPUs.
- Intel OpenCL runtime still showing known performance gap with OpenMP on Xeon platforms.
BabelStream: Dot

- For SYCL, OpenCL, CUDA and HIP, we implemented a global reduction by hand as they don’t have one built in.
- Do see some performance loss in the SYCL version compared to what is possible on the platforms.
- SYCL performance matches underlying implementations in most cases.

IWOCL / SYCLcon 2020
BabelStream: Copy

- Memory copy kernel, with no floating point operations.
- Heat application should behave similarly to this kernel.
- See good and consistent performance on all the GPUs.
- Observe large range of performance on the Xeon CPU.
Heat: average performance

- Two SYCL versions:
  - 2D range:
    ```c
    parallel_for<...>(range<2>{n,n},...)
    acc[j][i]
    ```
  - 1D range:
    ```c
    parallel_for<...>(range<1>{n*n},...)
    acc[j+i*n]
    ```
- Consistent performance on NUC and AMD.
- Xeon performance mirrors that of BabelStream Copy.
- NVIDIA platform shows issues with underlying models, possibly driver related.
Heat: comparison to Copy

- Compare to performance of Copy as measured for each model.
- On Xeon see about 60% of attainable Copy bandwidth.
- Consistent performance on NUC.
- AMD shows high variability.
- This chart highlights the performance issues with CUDA and OpenCL on NVIDIA.

IWOCL / SYCLcon 2020
CloverLeaf

- Chart shows runtime, lower is better.
- SYCL within 10% of OpenCL performance.
- Reduction cause of performance gap on NVIDIA.
- The OpenCL runtime needs improvement on Xeon in order to SYCL to achieve its potential as a parallel programming model of choice.

IWOCL / SYCLcon 2020
Summary

- Often possible to write SYCL applications that get good performance across a number of platforms.
- SYCL performance close to lower level model such as OpenCL.
- All the source code is available online, at our GitHub page.
- Widespread and robust support from all vendors is needed now to ensure SYCL is a success for the HPC community.