Performance Portability of OpenMP on CPUs and GPUs

Dr Tom Deakin
University of Bristol, UK
Challenges at Exascale

- The coming generation of Exascale supercomputers will contain a diverse range of architectures at massive scale
  
  - **Perlmutter**: AMD EYPC CPUs and NVIDIA GPUs (pre-Exascale)
  - **Frontier**: AMD EPYC CPUs and Radeon GPUs
  - **Aurora**: Intel Xeon CPUs and Xe GPUs
  - **El Capitan**: AMD EPYC CPUs and Radeon GPUs
  - **Fugaku**: Fujitsu A64fx Arm CPUs
OpenMP

- Shared memory parallel programming in C, C++ and Fortran
- Cross-vendor industry standard led by OpenMP Architecture Review Board
- Uses compiler directives, along with API calls
- First version released in 1997
- Target model introduced in OpenMP 4.0 in 2013
  - Key refinements in OpenMP 4.5, released 2015
  - Mechanisms for parallelism and data movement between the host and a target device (i.e. a GPU)
  - Data transfer happens as combination of implicit rules and explicit directives written by programmer
OpenMP is one option for programming CPUs and GPUs using open standards in a performance portable way.
BabelStream

• BabelStream benchmark written to measure achievable (main) memory bandwidth
• Based on McCalpin STREAM benchmark, but with a number of key differences:
  – Arrays allocated on the heap
  – Problem size known only at runtime
  – Range of programming models to widen support for CPUs and GPUs
• Constructed of simple vector operations:
  – Copy: \( c[i] = a[i] \)
  – Mul: \( b[i] = \text{scalar} \times c[i] \)
  – Add: \( c[i] = a[i] + b[i] \)
  – Triad: \( a[i] = b[i] + \text{scalar} \times c[i] \)
  – Dot: \( \text{sum} += a[i] \times b[i] \)

http://hpc.tomdeakin.com
https://github.com/UoB-HPC/BabelStream
OpenMP Triad: CPU

#pragma omp parallel for
for (int i = 0; i < array_size; i++)
{
    a[i] = b[i] + scalar * c[i];
}
OpenMP Triad: GPU

```c
#pragma omp target enter data \
    map(to: a[0:array_size], b[0:array_size], c[0:array_size])

#pragma omp target teams distribute parallel for simd
for (int i = 0; i < array_size; i++)
{
    a[i] = b[i] + scalar * c[i];
}

#pragma omp target exit data \
    map(from: a[0:array_size], b[0:array_size], c[0:array_size])
```
Results

- Showing percentage of peak memory bandwidth
- Latest and greatest CPUs and GPUs from all vendors
- Complete coverage for OpenMP
  - Near complete coverage for other models like SYCL and Kokkos
- Generally see consistent good performance on all architectures, CPUs and GPUs

<table>
<thead>
<tr>
<th></th>
<th>OpenMP</th>
<th>Kokkos</th>
<th>OpenACC</th>
<th>CUDA</th>
<th>OpenCL</th>
<th>SYCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade Lake</td>
<td>77%</td>
<td>58%</td>
<td>24%</td>
<td>X</td>
<td>35%</td>
<td>36%</td>
</tr>
<tr>
<td>Skylake</td>
<td>81%</td>
<td>50%</td>
<td>28%</td>
<td>X</td>
<td>44%</td>
<td>43%</td>
</tr>
<tr>
<td>Knights Landing</td>
<td>91%</td>
<td>94%</td>
<td>60%</td>
<td>X</td>
<td>50%</td>
<td>54%</td>
</tr>
<tr>
<td>Rome</td>
<td>79%</td>
<td>118%</td>
<td>40%</td>
<td>X</td>
<td>16%</td>
<td>30%</td>
</tr>
<tr>
<td>Power 9</td>
<td>66%</td>
<td>77%</td>
<td>47%</td>
<td>X</td>
<td>59%</td>
<td>X</td>
</tr>
<tr>
<td>ThunderX2</td>
<td>80%</td>
<td>76%</td>
<td>X</td>
<td>X</td>
<td>32%</td>
<td>79%</td>
</tr>
<tr>
<td>Graviton 2</td>
<td>84%</td>
<td>82%</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>26%</td>
</tr>
<tr>
<td>A64FX P100</td>
<td>79%</td>
<td>58%</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>14%</td>
</tr>
<tr>
<td>V100</td>
<td>75%</td>
<td>76%</td>
<td>35%</td>
<td>31%</td>
<td>58%</td>
<td>77%</td>
</tr>
<tr>
<td>A100</td>
<td>88%</td>
<td>92%</td>
<td>92%</td>
<td>93%</td>
<td>X</td>
<td>86%</td>
</tr>
<tr>
<td>Turing</td>
<td>77%</td>
<td>86%</td>
<td>88%</td>
<td>88%</td>
<td>87%</td>
<td>87%</td>
</tr>
<tr>
<td>Radeon VII MI50</td>
<td>87%</td>
<td>90%</td>
<td>90%</td>
<td>90%</td>
<td>86%</td>
<td>X</td>
</tr>
<tr>
<td>IrisPro Gen9</td>
<td>71%</td>
<td>69%</td>
<td>9%</td>
<td>X</td>
<td>30%</td>
<td>E</td>
</tr>
</tbody>
</table>

BabelStream Triad array size=2**25

Figure: Deakin, et al. *Tracking Performance Portability on the Yellow Brick Road to Exascale*, P3HPC’20.
OpenMP 5

• Improved reduction experience
  – Reduction result now automatically map(tofrom: res)
• Metadirectives
• Host fallback with OMP_TARGET_OFFLOAD
• Open question... wrangling the two memory spaces (allocators vs map)
Summary

• Possible to write performance portable code in OpenMP

• Robust support in compilers improving all the time
  – Seen concerted effort over last year:
  – LLVM, Intel, GCC, AOMP, XL, CCE, ...
  – [https://www.openmp.org/resources/openmp-compilers-tools/](https://www.openmp.org/resources/openmp-compilers-tools/)
Resources

• What programming model should I use? http://uob-hpc.github.io/2020/05/05/choosing-models.html

• SYCL Summer Session talk: A Comparison of Programming Models with SYCL: https://youtu.be/x-z0l858mMc

• Check out my OpenMP course on GitHub: https://github.com/UoB-HPC/openmp-for-cs

• Sign up for our tutorial at SC’20: https://sc20.supercomputing.org/presentation/?id=tut155&sess=sess237
For the OpenMP specification, tutorials, forum, reference guides, and links to other resources, visit www.openmp.org