OpenMP®

SC'20 Booth Talk Series

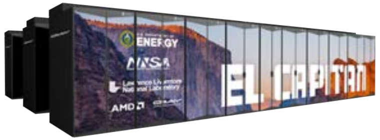
Performance Portability of OpenMP on CPUs and GPUs

Dr Tom Deakin
University of Bristol, UK

http://hpc.tomdeakin.com

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



The Next Platform, Jan 13th 2020: “HPC in 2020: compute engine diversity gets real”
https://www.nextplatform.com/2020/01/13/hpc-in-2020-compute-engine-diversity-gets-real/

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] = scalar * c[i]`
  - Add: `c[i] = a[i] + b[i]`
  - Triad: `a[i] = b[i] + scalar * c[i]`
  - Dot: `sum += a[i] * b[i]`

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

```
#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

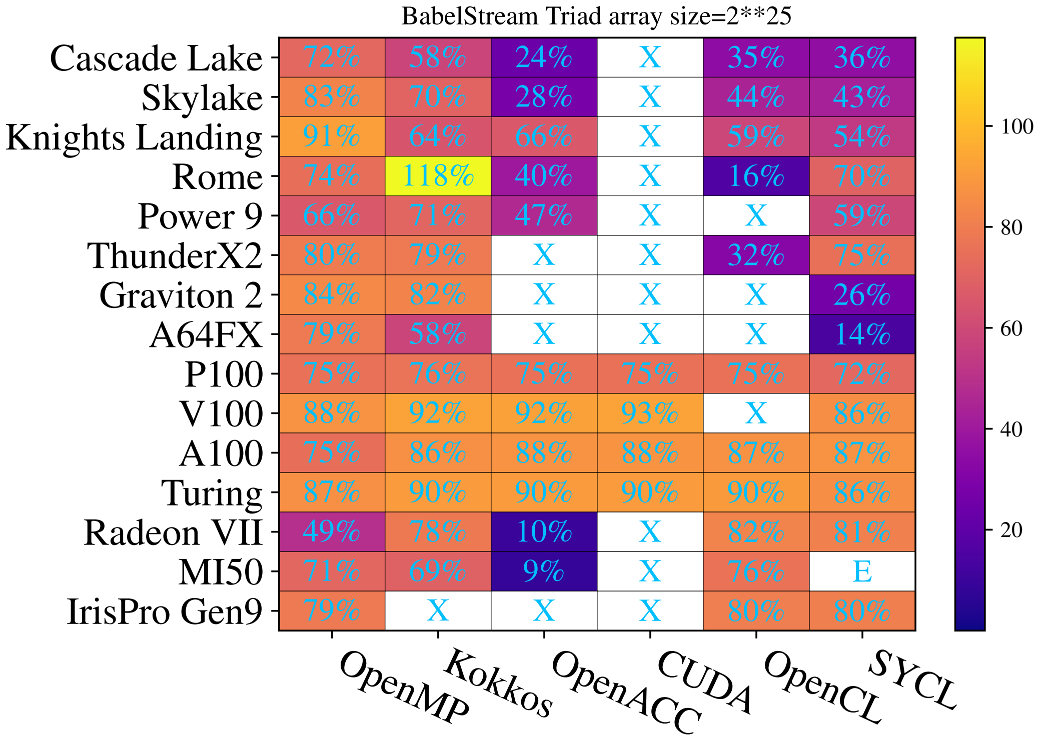


Figure: Deakin, et al. *Tracking Performance Portability on the Yellow Brick Road to Exascale*, P3HPC'20.

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/

University of BRISTOL

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-z0I858mMc
- 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

OpenMP®

SC'20 Booth Talk Series

For the OpenMP specification, tutorials, forum, reference guides, and links to other resources, visit **www.openmp.org**