Performance impact of formulating computations in SYCL on CPUs and GPUs

István Reguly, PPCU ITK

reguly.istvan@itk.ppke.hu

oneAPI DevSummit 2023

Performance & Portability with SYCL

- Goal of SYCL: single high-level C++ code, many target architectures
  - Portability
  - Developer productivity
- A lot of work to make it possible
  - DPC++ compiler, OpenSYCL compiler
  - GPUs covered well (Intel, NVIDIA, AMD)
  - CPUs less so
- Performance portability?
  - Multiple ways of expressing computation
  - How well does it map to the hardware?
  - SIMT maps well to GPUs, not great to CPUs

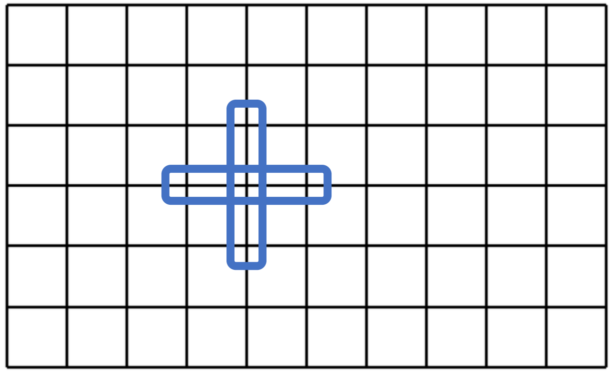
Simplest case – 2D/3D iteration space

```
Queue.submit([&](cl::sycl::handler &cgh) {
...//Accessors
  cgh.parallel_for<class apply_stencil>(
        cl::sycl::range<2>(jmax, imax),
        [=](cl::sycl::item<2> idx) {
...//Views, bounds checking

    Anew(0,0) = 0.25f *
           ( A(1,0) + A(-1,0)
           + A(0,-1) + A(0,1));
```

"**Flat**" parallel loop: just specify how many work items per dimension are required

Expressed from viewpoint of "current" gridpoint, with relative offsets



- Simplest way to do things – does not prescribe anything about how work items should be grouped or mapped to hardware

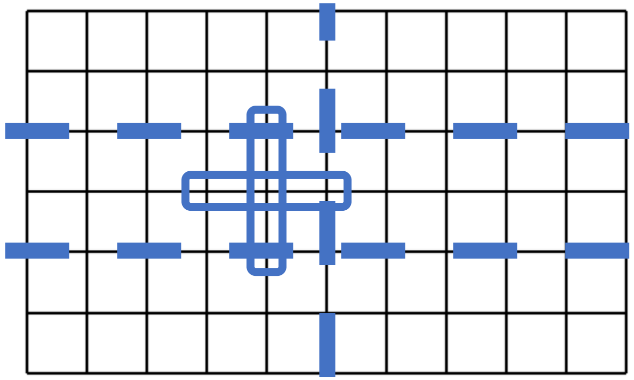
Look of a stencil loop – nd_range

```
Queue.submit([&](cl::sycl::handler &cgh) {
...//Accessors
  cgh.parallel_for<class apply_stencil>(
       cl::sycl::nd_range<2>(
               cl::sycl::range<2>(jmax, imax),
               cl::sycl::range<2>(4,    32)),
       [=](cl::sycl::nd_item<2> idx) {
...//Views, bounds checking

    Anew(0,0) = 0.25f *
           ( A(1,0) + A(-1,0)
           + A(0,-1) + A(0,1));
```

"**nd_range**" parallel loop: just specify how many work items per dimension and split them into workgroups

Expressed from viewpoint of "current" gridpoint, with relative offsets

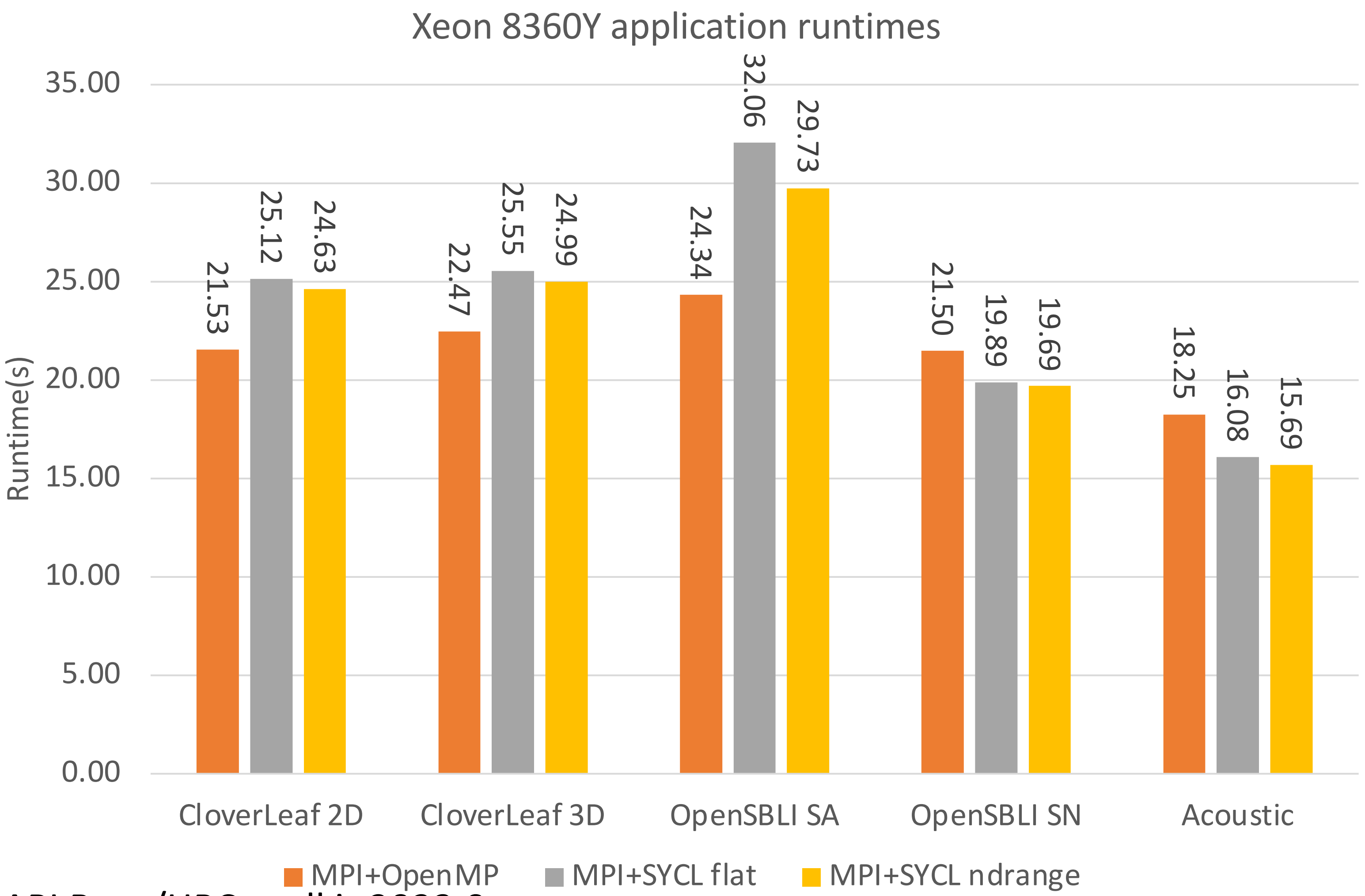


- Explicit way of grouping work items and mapping them to hardware – can control/impact cache locality. But entirely up to the programmer!
- Flat still has to map to this – but runtime gets to decide how (what sizes/granularity)

Structured-mesh stencil codes

- Boilerplate generated by OPS
  - Pure MPI, MPI+OpenMP, MPI+SYCL (CUDA/HIP/OpenCL/OpenACC)
- CloverLeaf 2D/3D – 7680^2, 408^3
  - Hydrodynamics code from AWE, part of Mantevo suite
  - ~100 nested loops, ~35 doubles per grid point, lots of small boundary loops
- OpenSBLI – Shock Boundary Layer Interactions - 320^3
  - Finite Difference Navier-Stokes solver with shock capturing from Univ of Southampton
  - High-level pyhton interface, generates OPS
  - Two versions – Store All (SA) or Store None (SN) – how much recompute for derivatives
- Acoustic solver – 320^3
  - 8th order finite differences
  - Very costly MPI halo exchanges
  - Two variants

Performance on Intel Ice Lake + DPC++

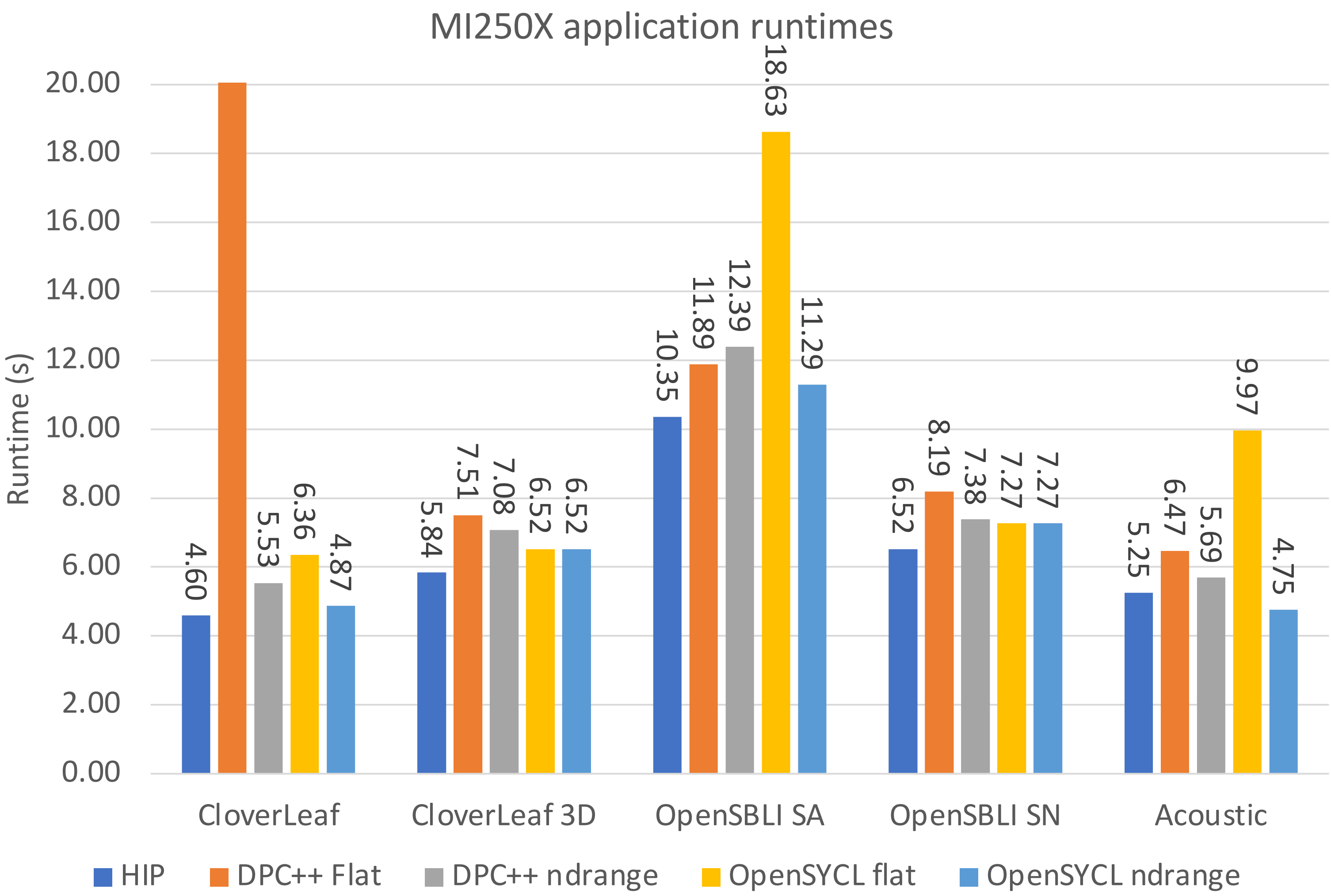


OneAPI Base/HPC toolkit 2023.0

DPC++ on CPUs vs. OpenMP

- More scheduling overhead
- Better vectorization
- Flat uses good guess for WG size
- nd_range:
  - `get_global_id(0)` vs. `get_global_id()[0]`

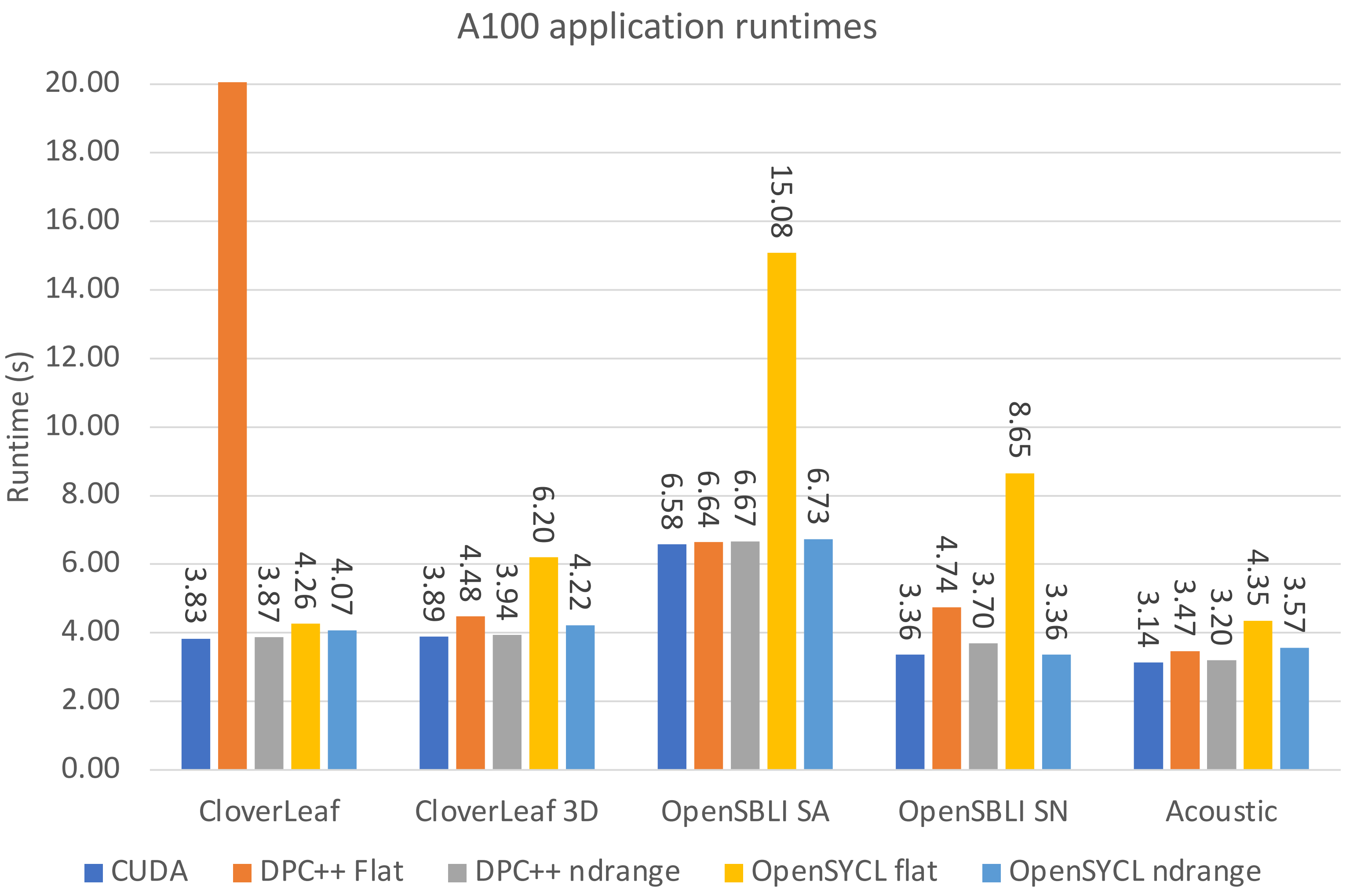
Performance on AMD MI250X (1GCD)



DPC++ and OpenSYCL

- Very close to HIP
- Flat sometimes very bad WG size
- nd_range:
  - Tuned size - up to 30% vs “reasonable default”
- Higher SGPR use
- More INT instructions
- Lower occupancy
  - But better cache performance

Performance on NVIDIA A100

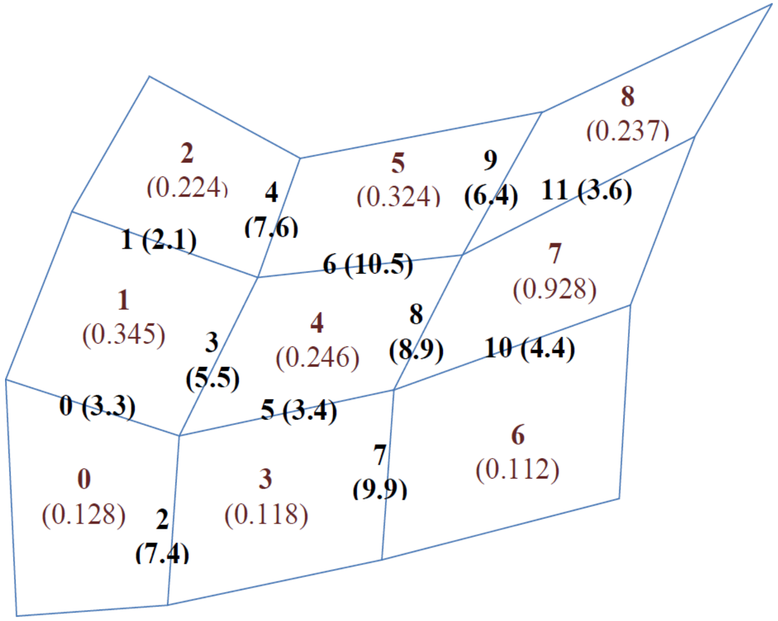


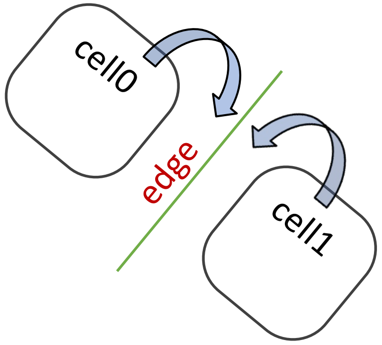
DPC++ and OpenSYCL

- Very close to CUDA
- DPC++ better choice of flat WG sizes
- nd_range:
  - Tuned size - up to 30% vs “reasonable default”, but lot less than MI250X

Unstructured mesh applications

- More complex computations
- Data-driven dependencies
- Common pattern of computation:
  - Loop over cells, compute something
  - Add/subtract to data on connected edges
- Parallel execution has to consider race conditions
- Data reuse & spatial/temporal locality non-trivial

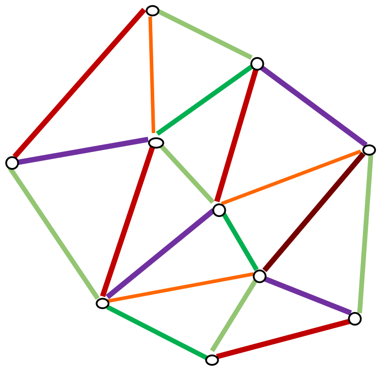




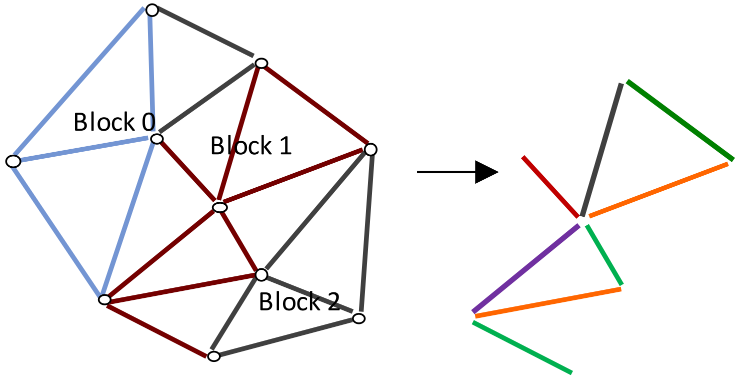
Unstructured mesh execution strategies

- Sequential execution – edge after edge, updating vertices. Good data locality
- Resolving race conflicts in shared memory parallel environments
  - Global coloring
    - Very simple, no data locality
  - Hierarchical coloring
    - Parallelism between blocks, sequential/parallel within block
    - Data locality within blocks, not across
  - Atomics (fp64)
    - Limited support/performance
    - Good data locality
- Not all can be used for different hardware, performance varies too
- No performance portability…

Global coloring

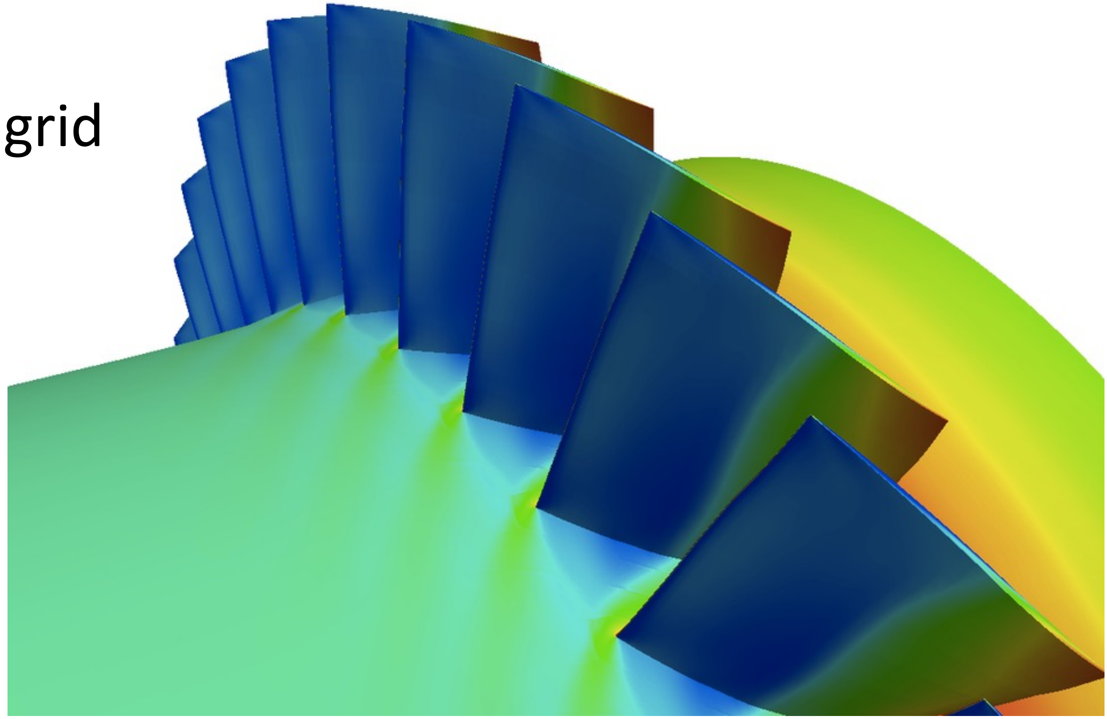


Hierarchical coloring



MG-CFD – OP2 UNSTRUCTURED MESH

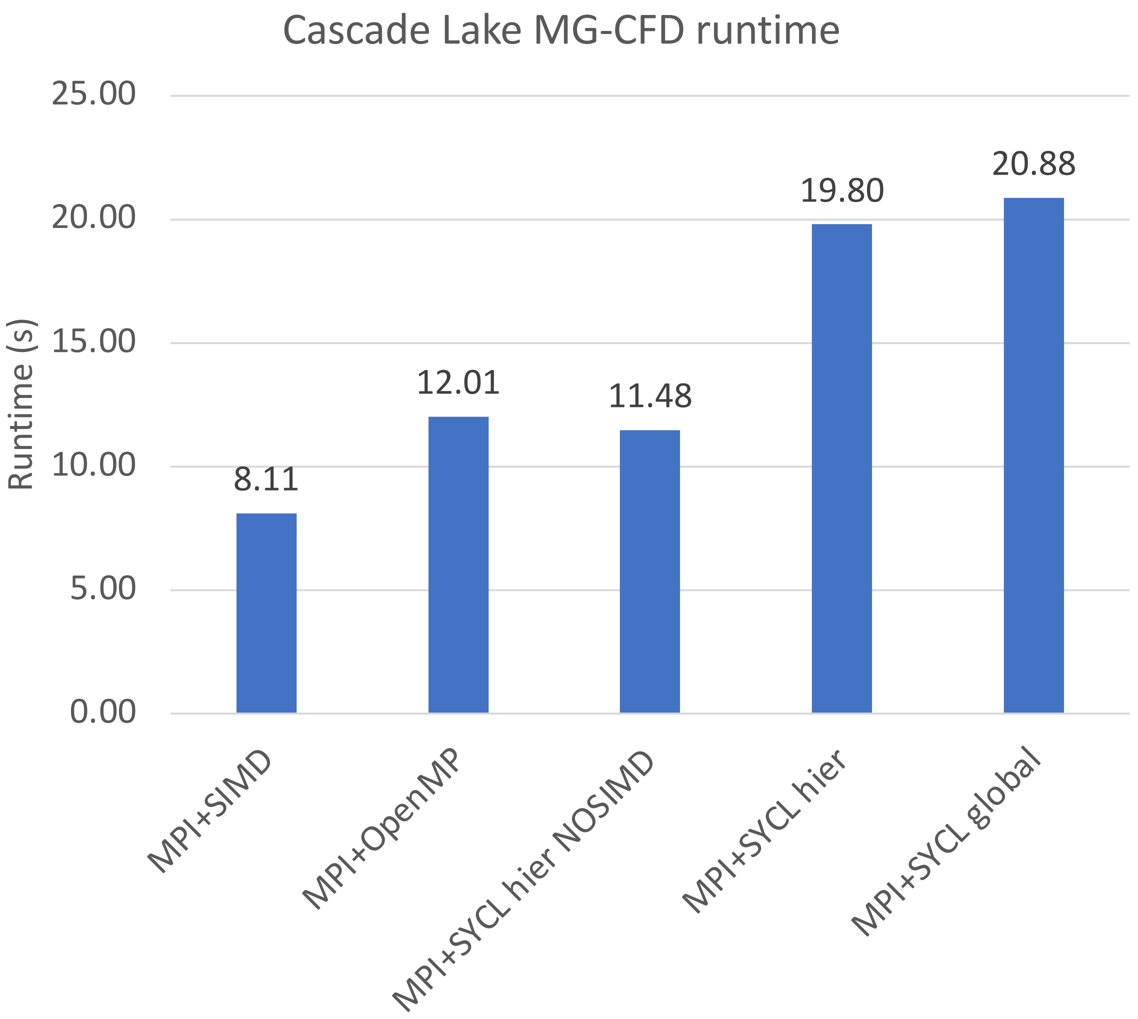
- MG-CFD: A proxy application of Rolls-Royce Hydra
  - Open-source multigrid unstructured mesh mini-application – easy to use, modify, distribute
  - Capture key performance characteristics:
    CFD computation, FV, unstructured mesh, multigrid
  - NASA Rotor37, 4 multigrid levels,
    8M vertices on finest level



https://github.com/warwick-hpsc/MG-CFD-app-[plain, OP2]

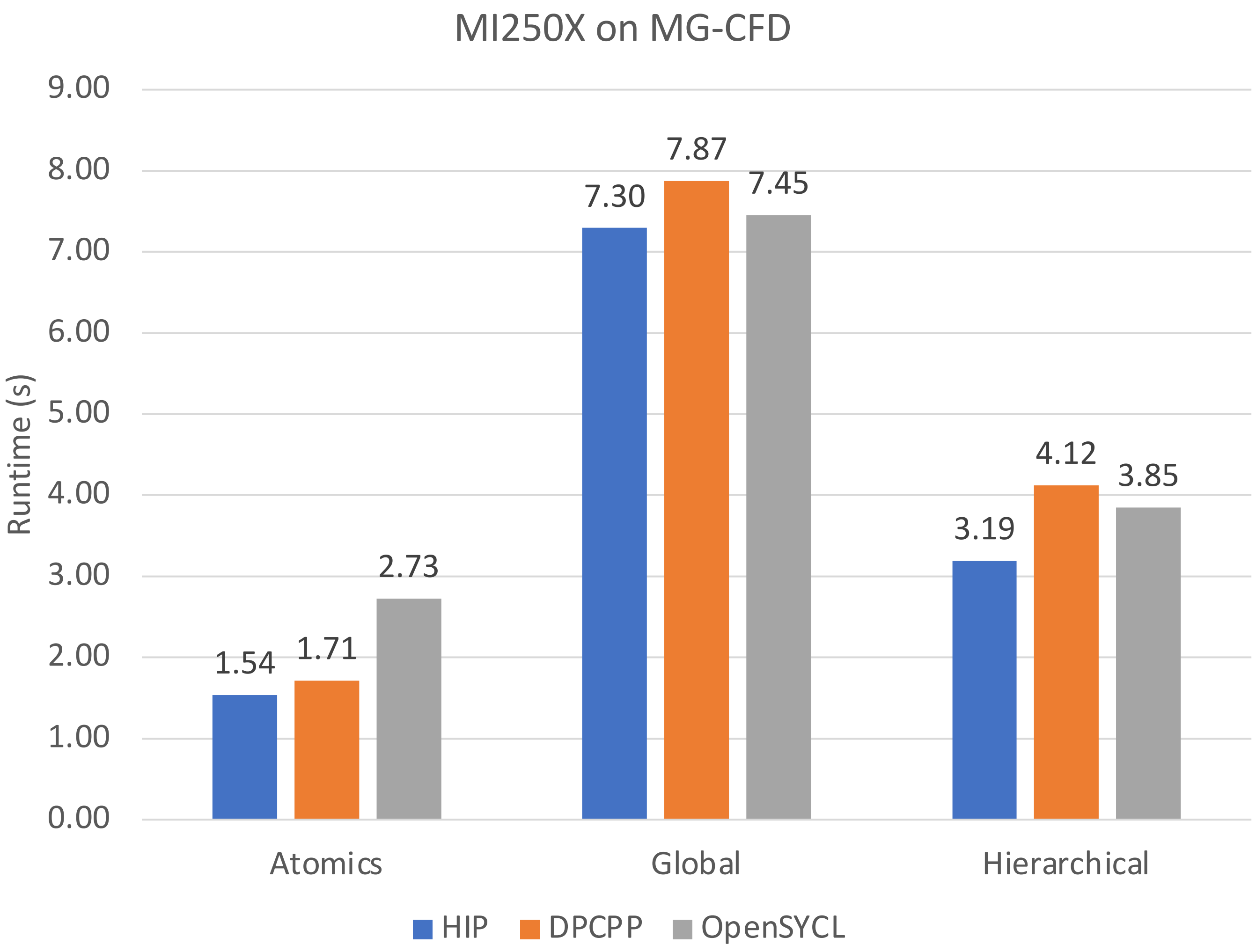
Owenson A.M.B., Wright S.A., Bunt R.A., Ho Y.K., Street M.J., and Jarvis S.A. (2019), An Unstructured CFD Mini-Application for the Performance Prediction of a Production CFD Code, Concurrency Computat: Pract Exper., 2019

Performance on Cascade Lake CPU



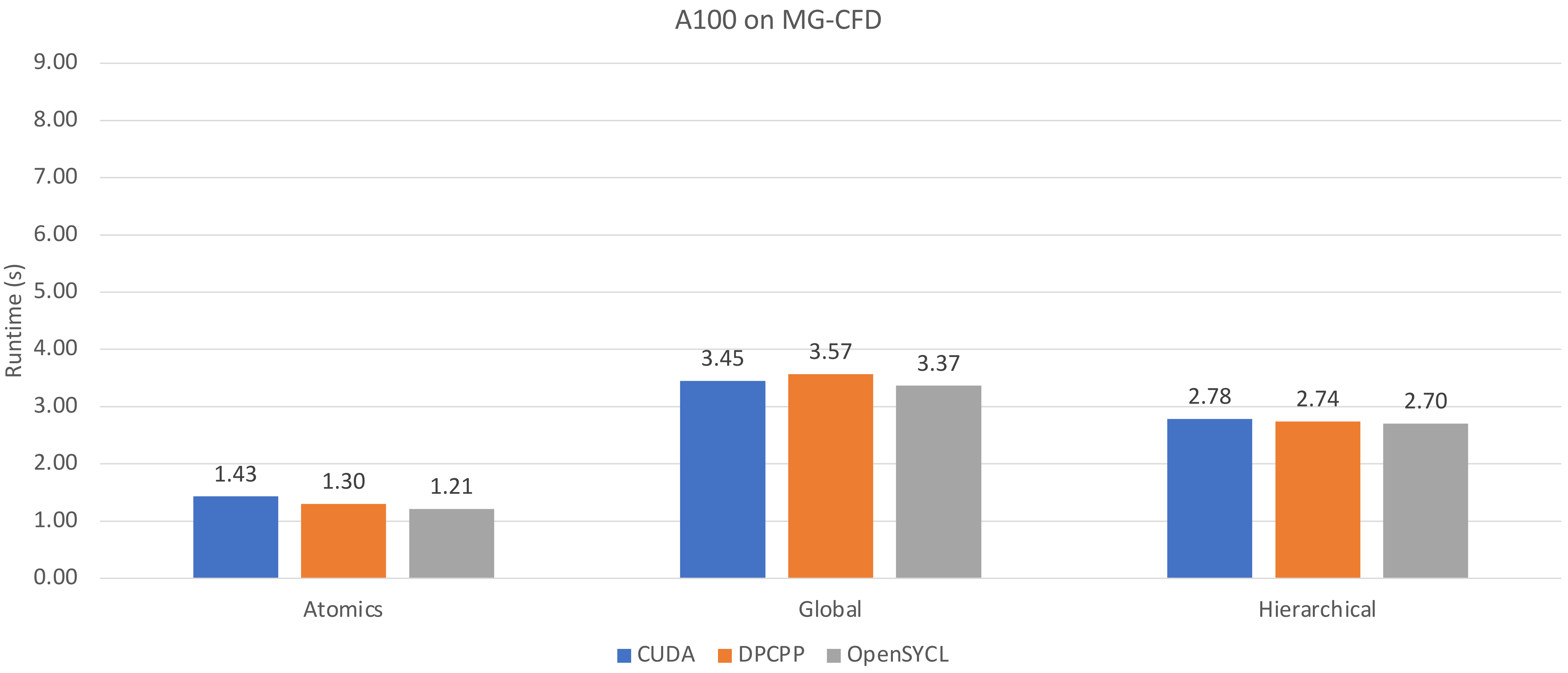
- MPI+SIMD: “ideal” performance
  - Explicit vector pack/unpack
- OpenMP
  - Further loss of data locality
  - No vectorization
- SYCL
  - Hier NOSIMD: matches OpenMP
  - Global: vectorizes, but poor locality
  - Hier: vectorizes, but…

Performance on AMD MI250X (1 GCD)



- OpenSYCL had to use “safe” atomics, HIP/DPC++ unsafe
- Atomic:
  - 3500 byte/wave read
  - 800 byte/wave write
  - 91% cache hit in L2
- Global:
  - 39000 byte/wave read
  - 8500 byte/wave write
  - 58% cache hit in L2
- Hierarchical:
  - 8600 byte/wave read
  - 2700 byte/wave write
  - 83% cache hit in L2

Performance on NVIDIA A100



Conclusions

- Flat vs. nd_range formulation: good guess by runtime (bad from user!)
- Key challenge: understanding mapping from SYCL code (SIMT abstraction) to the hardware
  - Reasonably trivial for GPU architectures, where the hardware is a good fit for SIMT
  - Still problematic for SIMD architectures (such as CPUs)
    - oneAPI is quite aggressive about vectorization, and the sub-group API really helps with mapping to SIMD. Performance getting quite close
- SYCL a much more productive alternative to OpenCL, and performance is improving rapidly
  - But the challenges in terms of performance portability remain
- Thanks to Intel for help through the oneAPI Innovator program