OpenMP®

SC23 Booth Talk Series

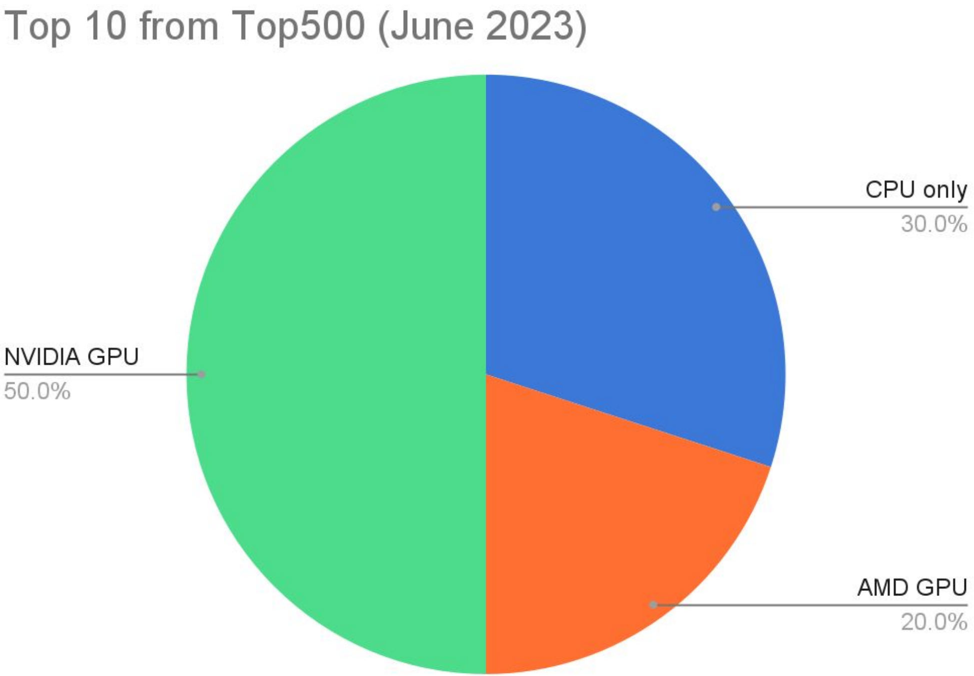
An evaluation of MPI+OpenMP on heterogeneous HPC systems

Wei-Chen (Tom) Lin
Tom Deakin
Simon McIntosh-Smith
University of Bristol

Aksel Alpay
Heidelberg University

Gonzalo Brito
NVIDIA

Background


Top 10 from Top500 (June 2023)
CPU only
30.0%
NVIDIA GPU
50.0%
AMD GPU
20.0%

Background

- Directives
  - **OpenMP**
  - OpenACC
- C/C++ (single-source)
  - SYCL
  - StdPar
  - Kokkos/RAJA
- C/C++ (multi-source)
  - OpenCL
- Vendor-specific
  - CUDA
  - HIP

Overview

- MPI+OpenMP
- Evaluation strategy
- Mini-apps introduction
- Results
- Conclusion

MPI+X on CPUs and GPUs

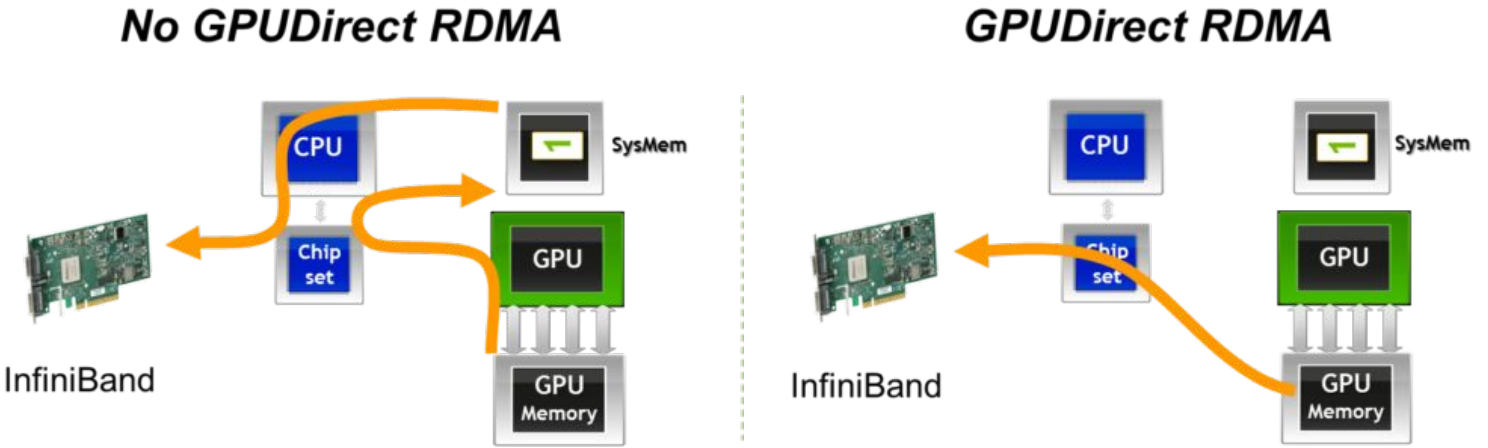


Image credit: Jiri Kraus - NVIDIA Technical Blog
https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/

MPI+OpenMP on CPUs and GPUs

```
auto d_a = static_cast<float*>(std::malloc(sizeof(float)
     * N);
std::fill(d_a, d_a + N, 42.0);
#pragma omp target enter data map(to:d_a[ : N])
#pragma omp target teams distribute parallel for simd
for (int i = 0; i < N; ++i) { d_a[i] *= 2; }
// Option 1: Device to host copy via data update
     directives
#pragma omp target update from(d_a[ : N])
 { MPI_Isend(d_a, N, MPI_FLOAT, ...); }
// Option 2: device-aware MPI
#pragma omp target data use_device_ptr(d_a)
{ MPI_Isend(d_a, N, MPI_FLOAT, ...); }
```

MPI+CUDA/HIP on CPUs and GPUs

```
__global__ void twice(float *a) { *a *= 2 };
// later
float h_a[] = { 42.0, ... };
float *d_a;
cudaMalloc(&d_a, sizeof(float));
cudaMemcpy(d_a, h_a, sizeof(float) * N,
     cudaMemcpyHostToDevice);
twice<<<1,1>>>(d_a); // Asynchronously execute on device

// Option 1: Device to Host copy via explicity API call
cudaMemcpyAsync(h_a, d_a, sizeof(float) * N,
     cudaMemcpyDeviceToHost);
cudaStreamSynchronize(0); // Block host thread on
     completion
MPI_Isend(h_a, N, MPI_FLOAT, ...); // Send results

// Option 2: Device-aware MPI with direct memory access
cudaStreamSynchronize(0);  // Block host thread on
     completion
MPI_Isend(d_a, N, MPI_FLOAT, ...); // Directly send
     results
```

MPI+Kokkos on CPUs and GPUs

```
Kokkos::View<float*> d_a("d_a", n);
auto h_a = Kokkos::create_mirror_view(d_a);
for (int i = 0; i < N; ++i) { h_a(i) = 42.f; }
Kokkos::deep_copy(d_a, h_a);
Kokkos::parallel_for(N, KOKKOS_LAMBDA(int i) { d_a(i) *=
     2; });
// Option 1: Device to host copy via API call
Kokkos::deep_copy(h_a, d_a); // copy from host to device
MPI_Isend(h_a.data(), N, MPI_FLOAT, ...);
// Option 2: device-aware MPI/Unified Memory
Kokkos::fence(); // block host thread
MPI_Isend(d_a.data(), N, MPI_FLOAT, ...);
```

MPI+SYCL on CPUs and GPUs

```
using cl::sycl;
float h_a[] = { 42.f, ... };
queue queue;
buffer<float> d_a(h_a, N);
queue.submit([&](handler &h) {
  auto acc_a = d_a.get_access<access::mode::read_write>(h
      );
  h.parallel_for({N}, [=](auto idx) { acc_a[idx] *= 2.0;
      });
});
// Option 1: Device to host copy via accessors
auto acc = host_accessor<float, 1, access_mode::
     read_write>{d_a, N};
MPI_Isend(acc.get_pointer(), N, MPI_FLOAT, ...);
// Option 2a: device-aware MPI, SYCL 2020 conformant (
     Intel DPC++)
queue.submit([&](sycl::handler &h) {
  auto acc = acc_a.get_access<access_mode::read>(h);
  h.host_task([=, ...](sycl::interop_handle ih) {
    auto p = ih.get_native_mem<backend::ext_oneapi_cuda>(
         acc);
    MPI_Isend(p, N, MPI_FLOAT, ...);

  // Option 2b: Device-aware MPI, AdaptiveCpp-specific
  queue.submit([&](sycl::handler &h) {
    h.update(accessor{d_a, h, read_only});
  }).wait_and_throw();
  MPI_Isend(d_a.get_pointer(d), N, MPI_FLOAT, ...);
```

MPI+StdPar on CPUs and GPUs

```
// Setup host data
auto a = static_cast<float*>(std::malloc(sizeof(float) *
     N);
std::fill_n(a, N, 42.0);
// Define kernel and execute synchronosuly using the
     selected policy and iteration domain
std::for_each_n(std::execution::par_unseq,
  std::views::iota(0).begin(), N, [=](int i) { a[i] *=
     2.0; });
// Read host data; if kernel is executed on different
     address space, data movement is implementation
     defined
std::cout << "Result=" << a[0] << "\n";
```

Evaluation

- Baseline bandwidth:
  - BabelStream
- Mini-apps
  - CloverLeaf (Structured Grid)
  - TeaLeaf (SpMV)
- Platforms
  - NVIDIA, AMD, and Intel GPUs
  - X86 Intel, AMD CPUs
  - AArch64 AWS HPC CPUs

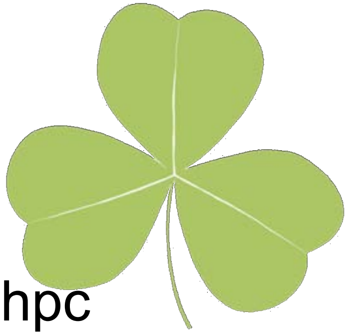
BabelStream

- Memory-bandwidth benchmark
  - Port of the McCalpin STREAM benchmark to models
- Source code available on GitHub
  - https://github.com/UoB-HPC/BabelStream

Algorithm 1 BabelStream kernels

procedure COPY($A[n], C[n], n$)
    for $i \leftarrow 0, n$ **do**
        $C[i] \leftarrow A[i]$
procedure MUL($A[n], B[n], C[n], scalar, n$)
    for $i \leftarrow 0, n$ **do**
        $B[i] \leftarrow scalar * C[i]$
procedure ADD($A[n], B[n], C[n], n$)
    for $i \leftarrow 0, n$ **do**
        $C[i] \leftarrow A[i] + B[i]$
procedure TRIAD($A[n], B[n], C[n], scalar, n$)
    for $i \leftarrow 0, n$ **do**
        $A[i] \leftarrow B[i] + (scalar * C[i])$
procedure DOT($A[n], B[n], scalar, n$)
    for $i \leftarrow 0, n$ **do**
        $R \leftarrow R + (A[i] * B[i])$
    return R

CloverLeaf



- Proxy application for 2D hydrodynamics, part of SPEChpc
- Source code available on GitHub
  - https://github.com/UoB-HPC/cloverleaf
  - Implementation in many programming models
- Mixed memory-bandwidth bound; measurements in total runtime
- Structured grid; stencil access pattern
- Reductions
- Complex application, >100 unique kernels + MPI halo exchange

TeaLeaf

- Proxy application for heat conduction, part of SPEChpc
- Source code available on GitHub
  - https://github.com/UoB-HPC/tealeaf
  - Implementation in many programming models
- Mixed memory-bandwidth bound; measurements in total runtime
- Structured grid; SpMV
- Reductions
- MPI halo exchange

Evaluation Setup - Hardware

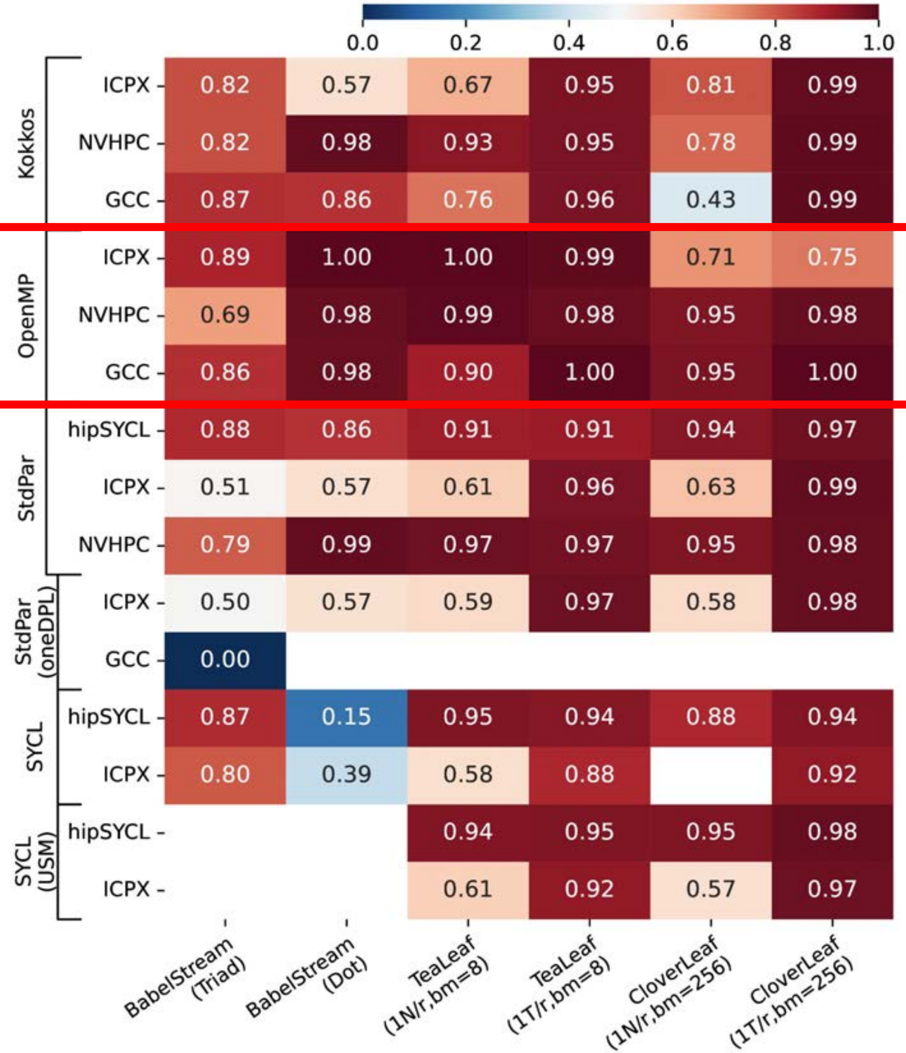
Vendor	Name	Architecture	Abbr.	Platform Topology	Interconnect	Total NUMA nodes	Single-node Theoretical Peak Mem. Bandwidth (GB/s)
Intel	Xeon Gold 6338	x86, Ice Lake	IceLake	8 nodes (32C*2)	IB HDR200	2 (1 per socket)	410
AMD	EPYC 7713	x86, Zen3 (Milan)	Milan	8 nodes (64C*2)	IB EDR100	8 (4 per socket)	410
AWS	Graviton 3e	AArch64, Neoverse V1	G3e	8 nodes (64C*1)	EFA (200Gbps)	1	307
NVIDIA	Tesla H100 (SXM 80GB)	Hopper	H100	2 nodes (4 GPUs)	IB NDR400	N/A	3,350
AMD	Instinct MI100	CDNA	MI100	2 nodes (4 GPUs)	IB EDR100	N/A	1,228
Intel	Data Center GPU Max 1550	Ponte Vecchio	PVC	1 node (4 GPUs*)	N/A	N/A	3,276

Evaluation Setup - Software&Benchmarks

Mini-app	Input deck	Grid size	Steps	Total Memory Requirement (GB)
TeaLeaf	BM5@2k	2000	CPU=2;GPU=4	0.49
	BM5@4k	4000	CPU=2;GPU=4	1.96
	BM5@8k	8000	CPU=2;GPU=4	7.92
CloverLeaf	BM16	3840	300	2.95
	BM64	7680	300	11.81
	BM256	15360	300	47.21

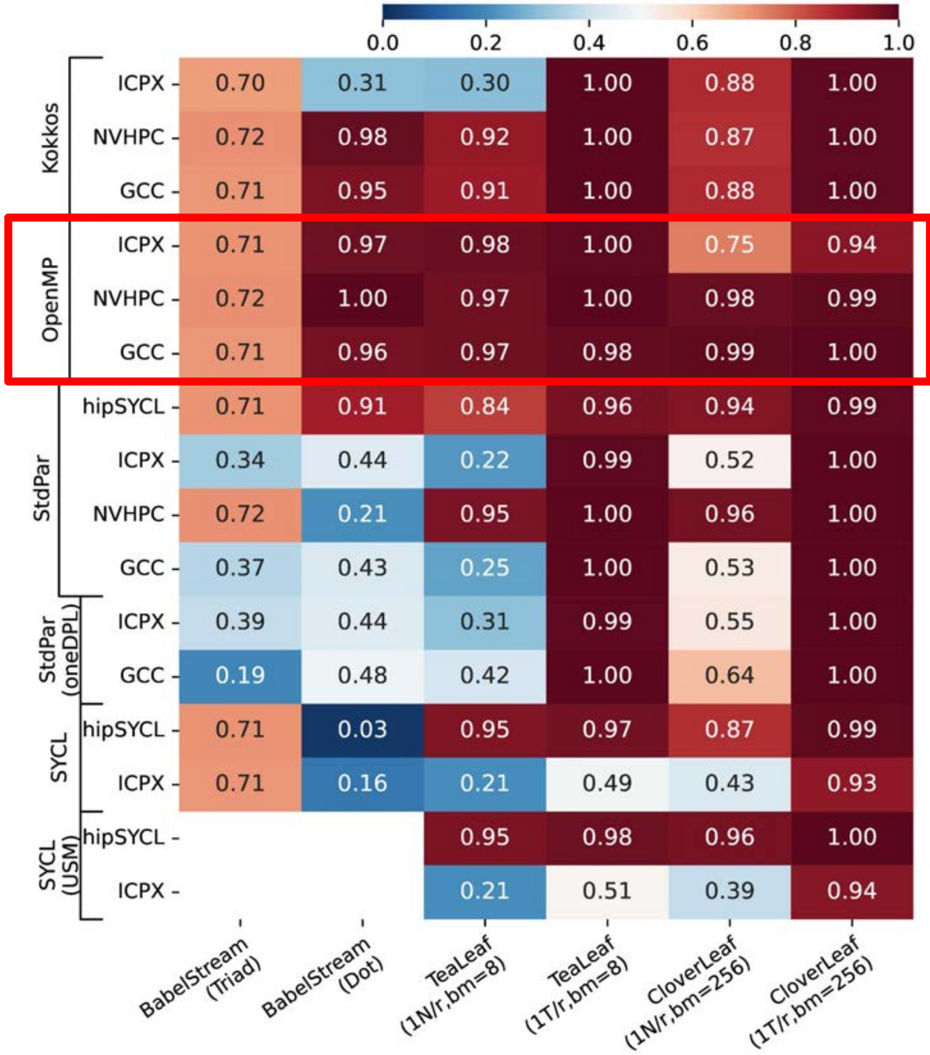
Compiler	MPI implementation	Compilers
Milan	Cray MPICH 8.1.25	GCC 13.1.0 oneAPI 2023.2 NVHPC 23.7 AdaptiveCpp 7b2e459
IceLake	Intel MPI 2021.10	
Graviton3e	AWS OpenMPI 4.1.5	GCC 13.1.0 ACfL 23.04 (armclang, LLVM 116) NVHPC 23.7 AdaptiveCpp 7b2e459
H100	HPC-X 2.15	NVHPC 23.7 oneAPI 2023.2 (w/ Codeplay plugin) AdaptiveCpp 7b2e459
MI100	Cray MPICH 8.1.25	ROCm 5.4.1 AOMP 16.0.3 oneAPI 2023.2 (w/ Codeplay plugin) AdaptiveCpp 7b2e459
PVC1550	Intel MPI 2021.10	oneAPI 2023.2 AdaptiveCpp 7b2e459

CPU Baseline: Compilers



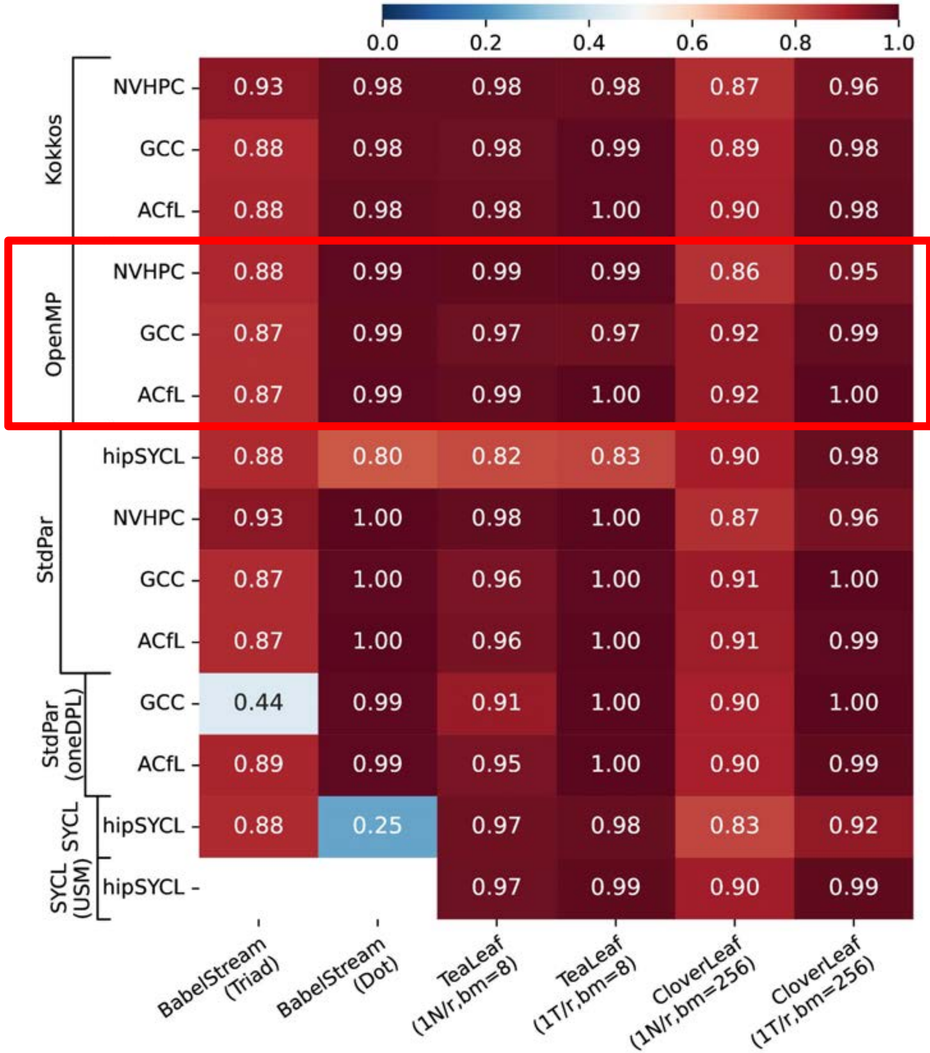
Intel IceLake

CPU Baseline: Compilers



AMD Milan

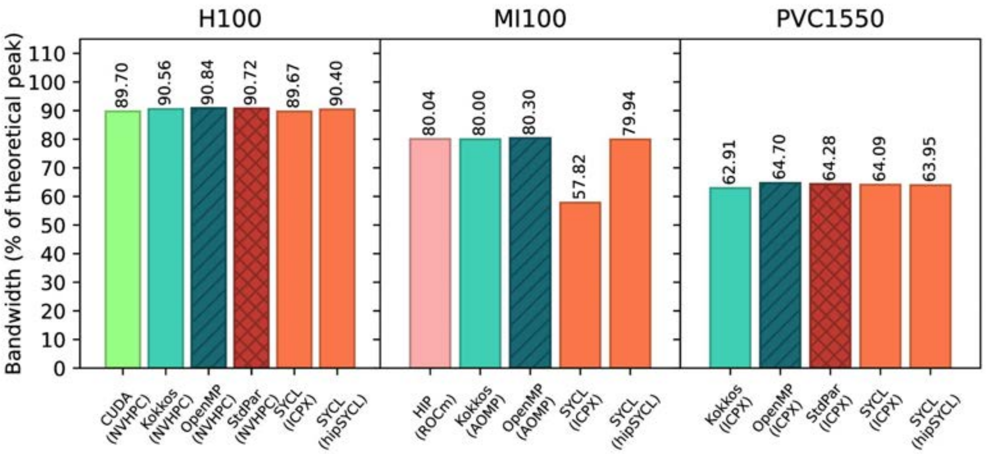
CPU Baseline: Compilers



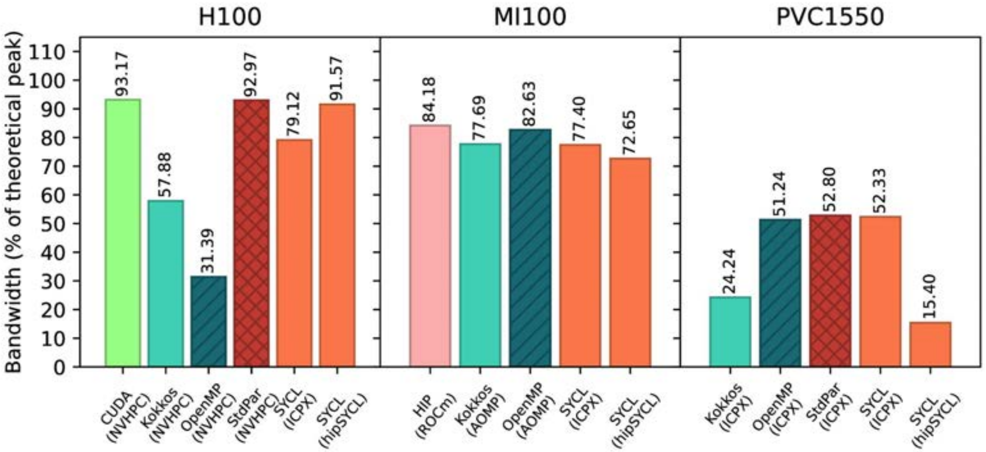
AWS Graviton 3e

GPU Baseline: BabelStream

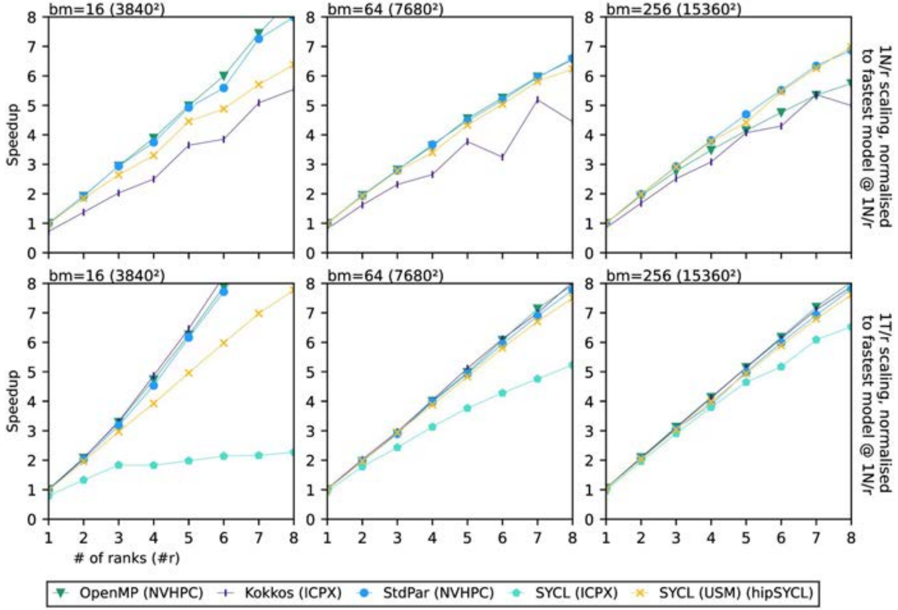
Triad Kernel



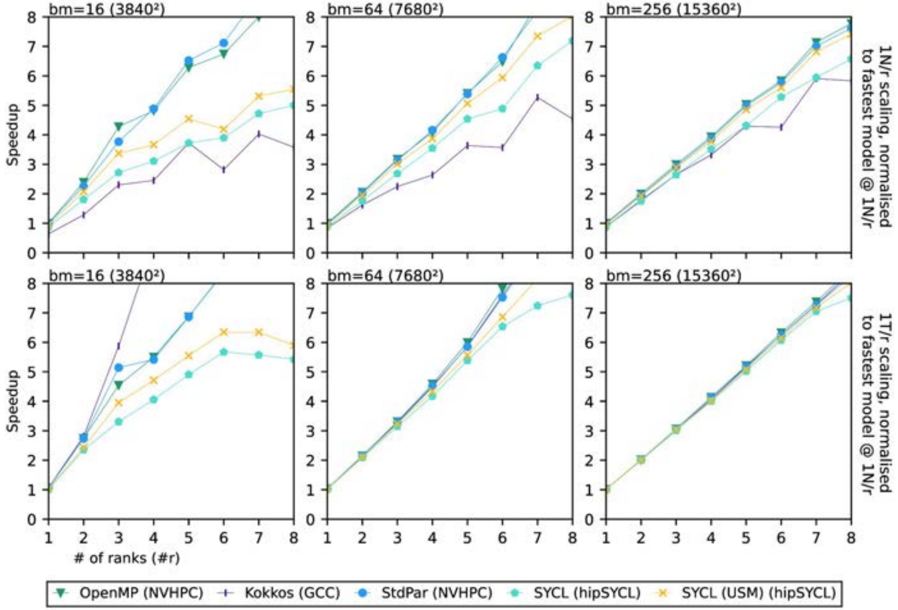
Dot kernel



Results: CloverLeaf (CPUs)

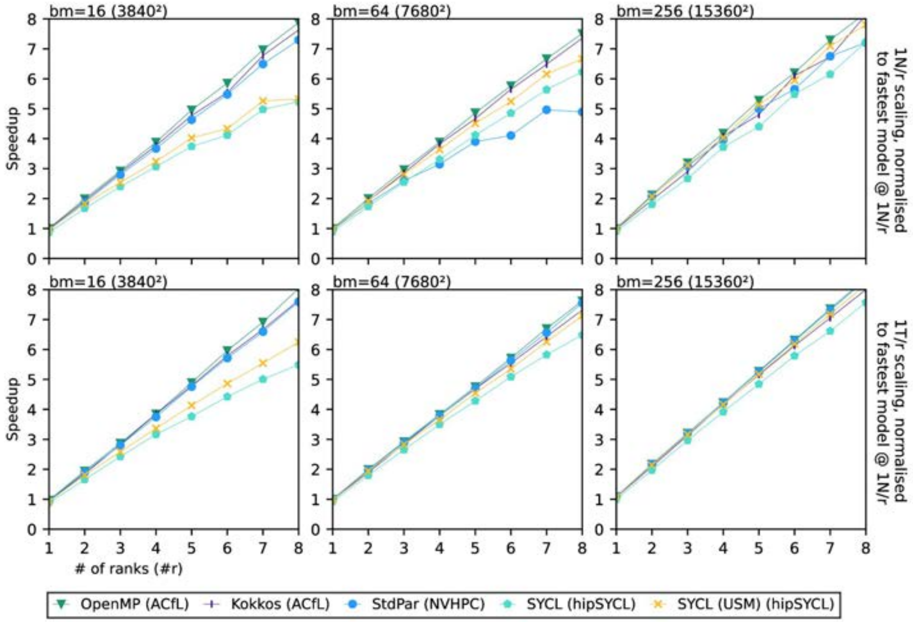


Intel IceLake



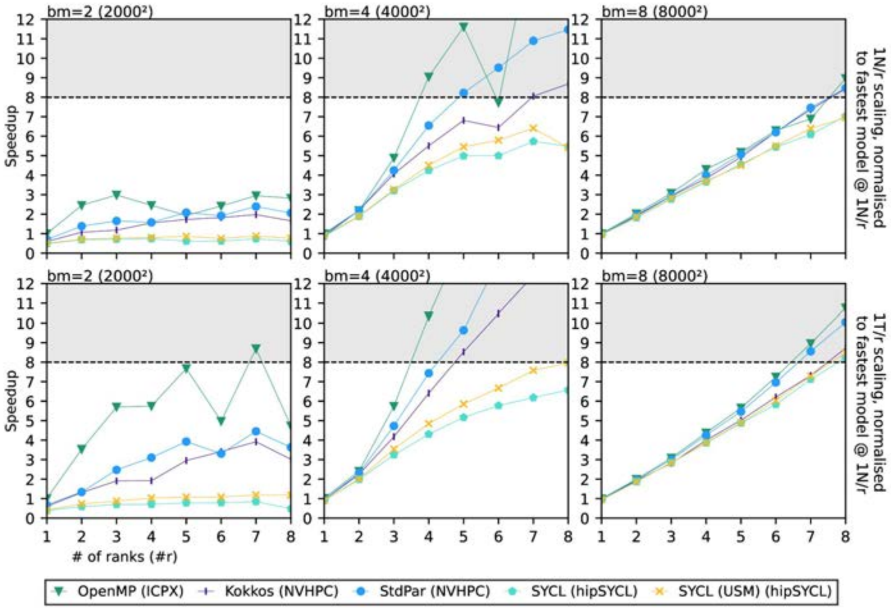
AMD Milan

Results: CloverLeaf (CPUs)

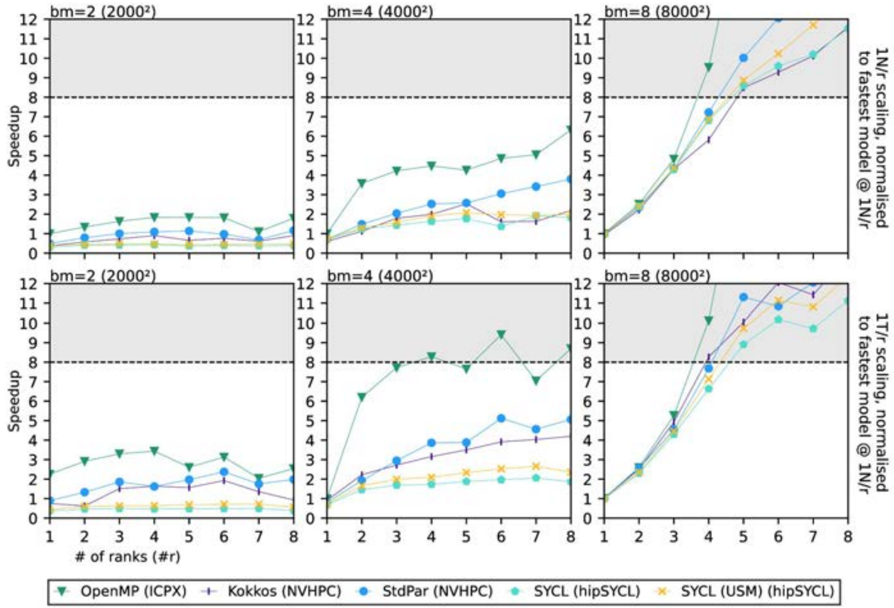


AWS Graviton 3e

Results: TeaLeaf (CPUs)

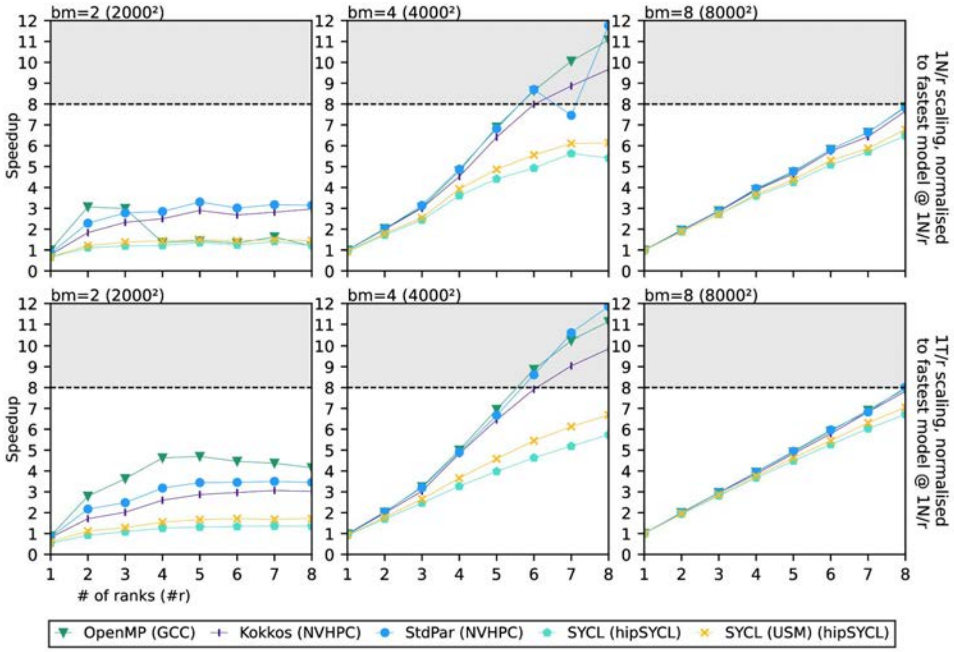


Intel IceLake



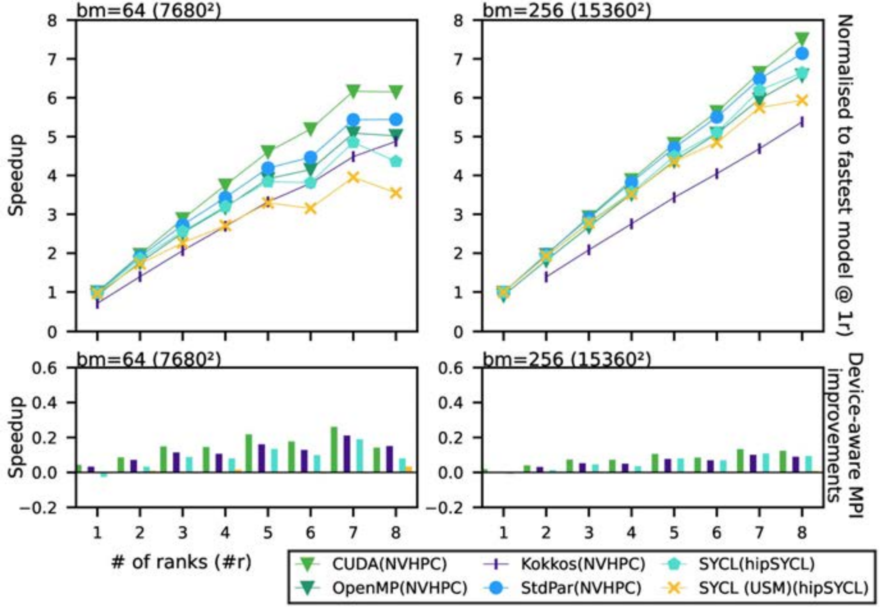
AMD Milan

Results: TeaLeaf (CPUs)

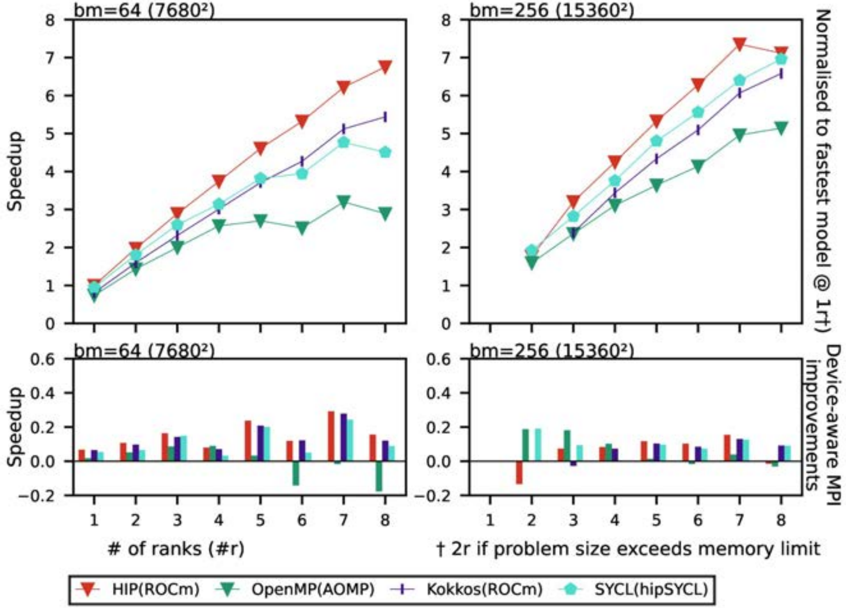


AWS Graviton 3e

Results: CloverLeaf (GPUs)

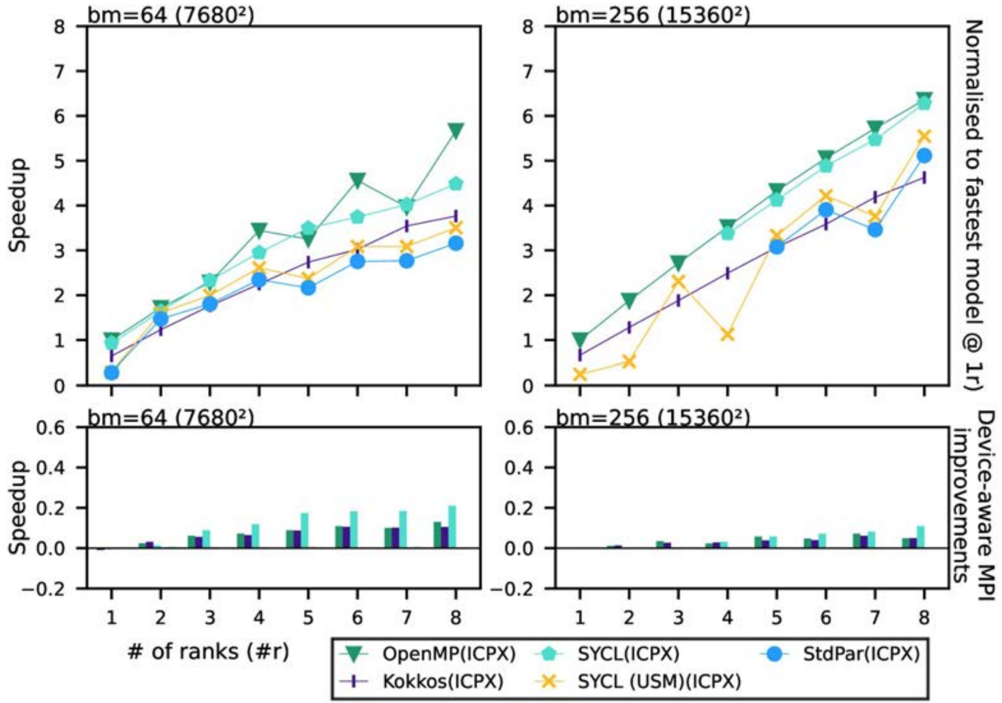


NVIDIA A100 (80GB)



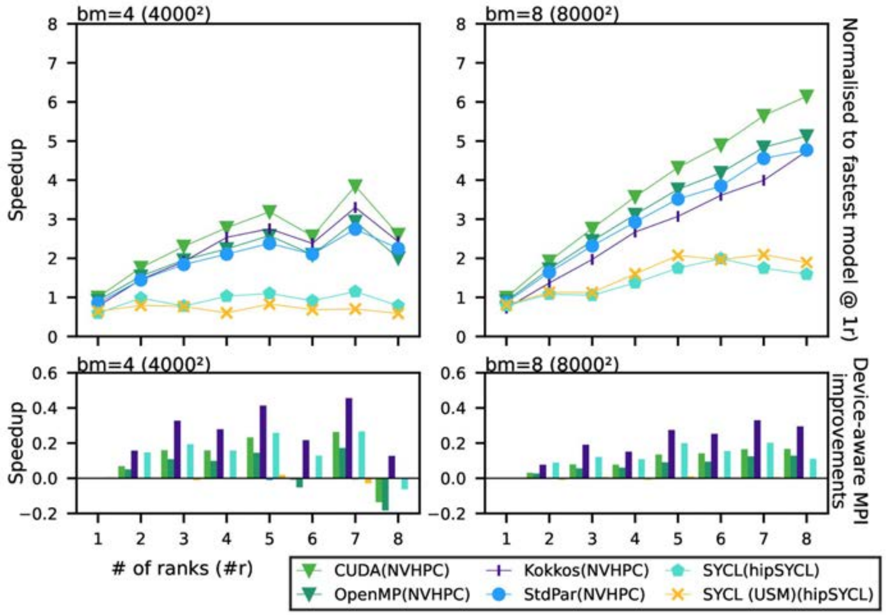
AMD MI100

Results: CloverLeaf (GPUs)

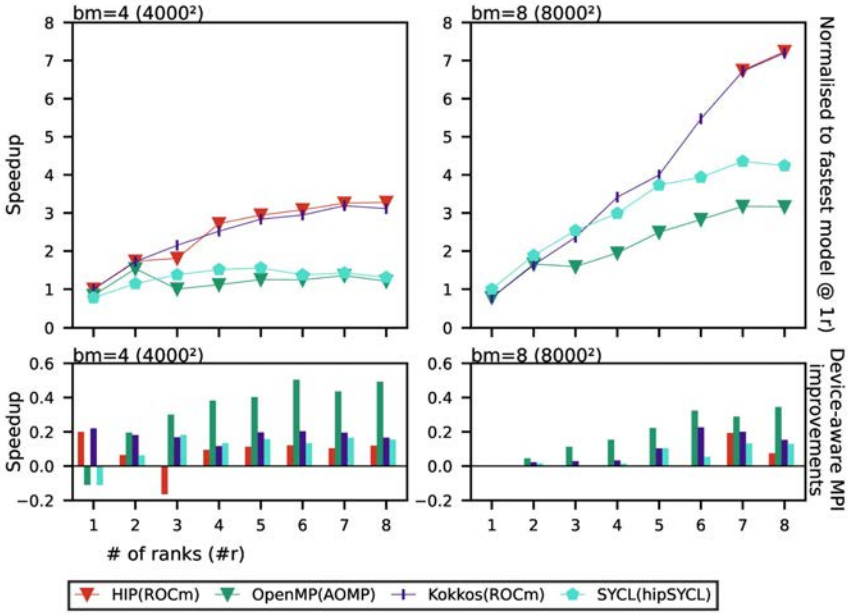


Intel PVC 1550

Results: TeaLeaf (GPUs)

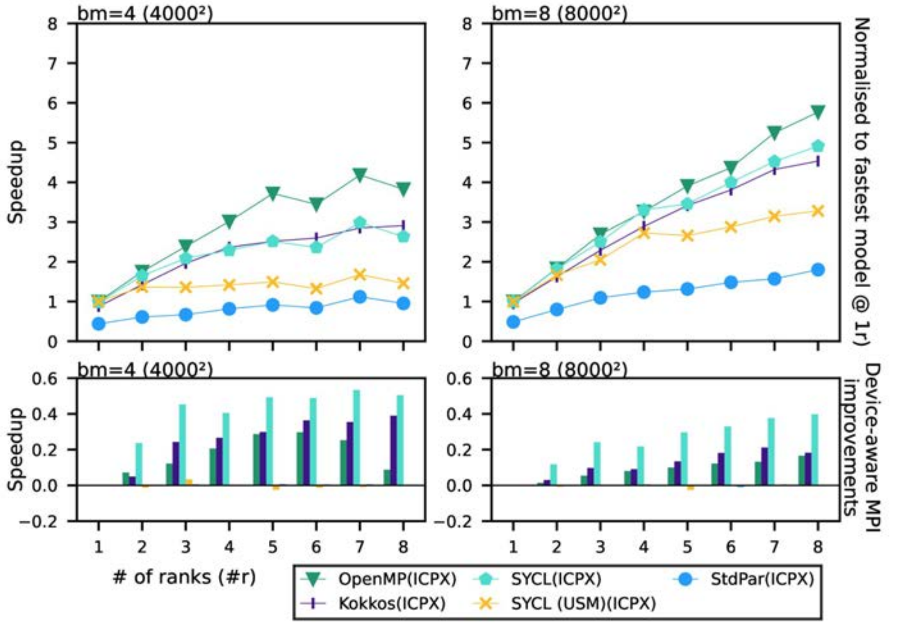


NVIDIA A100 (80GB)



AMD MI100

Results: TeaLeaf (GPUs)



Intel PVC 1550

Conclusion

- Performance portability for hybrid MPI+X applications achieved on only certain platforms with very specific compilers.
- Overall performance is not primarily dictated by the specific spelling/encoding of parallelism from each programming model, but how well the underlying implementation optimises for the hardware platform.
- Win for OpenMP on CPUs
- OpenMP GPU performance tied to vendor support
  - Excellent on NVIDIA/Intel
  - Poor on AMD due to MPI progress issues

University of BRISTOL

Acknowledgements

This work used the Isambard (http://gw4.ac.uk/isambard/) UK National Tier-2 HPC Service operated by GW4 and the UK Met Office, and funded by EPSRC (EP/P020224/1).
This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk). DiRAC is part of the National e-Infrastructure. The University of Bristol is an Intel oneAPI Center of Excellence, which helped support this work. This project has received funding from the European Union’s HE research and innovation programme under grant agreement No 101092877.
Some experiments in this presentation used resources from NVIDIA’s Selene and pre-EOS Supercomputers.
We are extremely grateful to AWS for supporting access to Graviton 3e.

OpenMP®

SC23 Booth Talk Series

openmp.org OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc23 OpenMP SC’23 booth talk videos and presentations