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)

- NVIDIA GPU: 50.0%
- CPU only: 30.0%
- 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

No GPUDirect RDMA

GPUDirect RDMA

Image credit: Jiri Kraus - NVIDIA Technical Blog
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, ...); }

bristol.ac.uk
MPI+CUDA/HIP on CPUs and GPUs

```c
__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 explicit 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_LAMBDACHR(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

```cpp
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);
    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

```cpp
// Setup host data
auto a = static_cast<float*>(std::malloc(sizeof(float) * N));
std::fill_n(a, N, 42.0);
// Define kernel and execute synchronously 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

1: procedure COPY(A[n], C[n], n)
2:    for i ← 0, n do
3:        C[i] ← A[i]
4: procedure MUL(A[n], B[n], C[n], scalar, n)
5:    for i ← 0, n do
6:        B[i] ← scalar * C[i]
7: procedure ADD(A[n], B[n], C[n], n)
8:    for i ← 0, n do
10: procedure TRIAD(A[n], B[n], C[n], scalar, n)
11:    for i ← 0, n do
12:        A[i] ← B[i] + (scalar * C[i])
13: procedure DOT(A[n], B[n], scalar, n)
14:    for i ← 0, n do
15:        R ← R + (A[i] * B[i])
16: return R
CloverLeaf

- Proxy application for 2D hydrodynamics, part of SPEChpc
- Source code available on GitHub
  - [https://github.com/UoB-HPC/cloverleaf](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](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

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

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

<table>
<thead>
<tr>
<th>Compiler</th>
<th>MPI implementation</th>
<th>Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milan</td>
<td>Cray MPICH 8.1.25</td>
<td>GCC 13.1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>oneAPI 2023.2</td>
</tr>
<tr>
<td>IceLake</td>
<td>Intel MPI 2021.10</td>
<td>NVHPC 23.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AdaptiveCpp 7b2e459</td>
</tr>
<tr>
<td>Graviton3e</td>
<td>AWS OpenMPI 4.1.5</td>
<td>GCC 13.1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACIFL 23.04 (armclang, LLVM 116)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NVHPC 23.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AdaptiveCpp 7b2e459</td>
</tr>
<tr>
<td>H100</td>
<td>HPC-X 2.15</td>
<td>NVHPC 23.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>oneAPI 2023.2 (w/ Codeplay plugin)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AdaptiveCpp 7b2e459</td>
</tr>
<tr>
<td>MI100</td>
<td>Cray MPICH 8.1.25</td>
<td>ROCm 5.4.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AOMP 16.0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>oneAPI 2023.2 (w/ Codeplay plugin)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AdaptiveCpp 7b2e459</td>
</tr>
<tr>
<td>PVC1550</td>
<td>Intel MPI 2021.10</td>
<td>oneAPI 2023.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AdaptiveCpp 7b2e459</td>
</tr>
</tbody>
</table>
## CPU Baseline: Compilers

<table>
<thead>
<tr>
<th></th>
<th>Kokkos</th>
<th>OpenMP</th>
<th>StdPar (oneDPL)</th>
<th>SYCL (USM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICPX</td>
<td>0.82</td>
<td>0.89</td>
<td>0.51</td>
<td>0.87</td>
</tr>
<tr>
<td>NVHPC</td>
<td>0.82</td>
<td>0.69</td>
<td>0.79</td>
<td>0.87</td>
</tr>
<tr>
<td>GCC</td>
<td>0.87</td>
<td>0.69</td>
<td>0.79</td>
<td>0.87</td>
</tr>
<tr>
<td>hipSYCL</td>
<td>0.88</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
</tr>
</tbody>
</table>

**bristol.ac.uk**

Intel IceLake
CPU Baseline: Compilers

bristol.ac.uk
CPU Baseline: Compilers

AWS Graviton 3e
### GPU Baseline: BabelStream

<table>
<thead>
<tr>
<th></th>
<th>H100</th>
<th>MI100</th>
<th>PVC1550</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Triad Kernel</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BabelStream</td>
<td>89.70</td>
<td>80.04</td>
<td>62.91</td>
</tr>
<tr>
<td>GPU Baseline</td>
<td>90.56</td>
<td>80.00</td>
<td>64.70</td>
</tr>
<tr>
<td>hip-opencl</td>
<td>90.84</td>
<td>80.30</td>
<td>64.28</td>
</tr>
<tr>
<td>cub</td>
<td>90.72</td>
<td>79.94</td>
<td>64.09</td>
</tr>
<tr>
<td>roc</td>
<td>89.67</td>
<td>79.58</td>
<td>63.95</td>
</tr>
<tr>
<td>OpenMP</td>
<td>90.40</td>
<td>79.41</td>
<td></td>
</tr>
<tr>
<td><strong>Dot kernel</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BabelStream</td>
<td>93.17</td>
<td>84.18</td>
<td>24.24</td>
</tr>
<tr>
<td>GPU Baseline</td>
<td>91.57</td>
<td>77.40</td>
<td>52.33</td>
</tr>
<tr>
<td>hip-opencl</td>
<td>92.97</td>
<td>82.63</td>
<td>52.80</td>
</tr>
<tr>
<td>cub</td>
<td>92.19</td>
<td>77.69</td>
<td>51.94</td>
</tr>
<tr>
<td>roc</td>
<td>91.12</td>
<td>77.40</td>
<td>51.40</td>
</tr>
<tr>
<td>OpenMP</td>
<td>91.57</td>
<td>72.45</td>
<td>51.40</td>
</tr>
</tbody>
</table>

[bristol.ac.uk](http://bristol.ac.uk)
Results: CloverLeaf (CPUs)

Intel IceLake

AMD Milan
Results: CloverLeaf (CPUs)

AWS Graviton 3e
Results: TeaLeaf (CPUs)

- Intel IceLake
- AMD Milan

bristol.ac.uk
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)

![Graphs showing speedup for different configurations of GPUs with 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
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.
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