Kokkos: State on Exascale Architectures

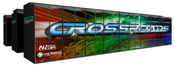
Daniel Arndt, Oak Ridge National Laboratory

Kokkos User Group Meeting 2023

December 12, 2023

The HPC Hardware Landscape

Primary Programming Models: OpenMP, Cuda, HIP, SYCL



LANL Crossroads
Intel CPUs,
OpenMP, SYCL?

LBNL Perlmutter
AMD CPU, NVIDIA GPU
CUDA

ORNL Frontier
AMD CPU, AMD GPU
HIP

ANL Aurora
Intel CPUs, Intel GPUs
SYCL

LLNL El Capitan
AMD CPU, AMD GPU
HIP

NVIDIA GPUs/CUDA

Supercomputer

- Summit(ORNL)
- Perlmutter(LBNL)
- Sierra(LLNL)
- Polaris(ANL)

Minimum version `Cuda` 11.0.0.

```
cmake ..\
  -DCMAKE_CXX_COMPILER=* \
  -DKokkos_ENABLE_CUDA=ON \
  -DKokkos_ARCH_NATIVE=ON \
  -DKokkos_ARCH_AMPERE80=ON
```

- Without compiler wrapper, `clang++` or `nvcc_wrapper` must be used.
- `nvhpc` only used as host compiler by default.

Cuda: Performance, A100

Results from `bytes_and_flops(TeamPolicy)`[1]

scalar	Bandwidth	Compute	Cache
`float`	1251 GiB/s	14280 GFlop/s	3762 GiB/s
`double`	1267 GiB/s	7592 GFlop/s	6938 GiB/s
`int32_t`	1222 GiB/s	18457 GFlop/s	4684 GiB/s
`int64_t`	1267 GiB/s	3778 GFlop/s	6895 GiB/s

- Peak FP64 Vector: 19.5 TFLOPS
- Memory Bandwidth: 1.6 TB/sec
- Cache Size: L1/L2: 192KB (per SM)/40 MB

[1]This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

AMD GPUs/HIP

Supercomputer

- Frontier(ORNL)
- El Capitan(LLNL)

Unsupported features

- `Tasks`

Minimum version `ROCm` 5.2.0.

```
cmake ..\
  -DCMAKE_CXX_COMPILER=hipcc \
  -DKokkos_ENABLE_HIP=ON \
  -DKokkos_ARCH_NATIVE=ON \
  -DKokkos_ARCH_AMD_GFX90A=ON
```

HIP: Performance MI250, one GCD

Results from `bytes_and_flops` (`TeamPolicy`)[2]

scalar	Bandwidth	Compute	Cache
`float`	1160 GiB/s	20544 GFlop/s	2756 GiB/s
`double`	1140 GiB/s	19320 GFlop/s	2883 GiB/s
`int32_t`	1150 GiB/s	20194 GFlop/s	2757 GiB/s
`int64_t`	1140 GiB/s	4979 GFlop/s	2865 GiB/s

- Peak FP64 Vector: 23.95 TFLOPS
- Memory Bandwidth: 1.6 TB/sec
- Cache Size: L1/L2: 16KB (per CU)/16 MB

[2]This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

HIP: Caveats

When using `HIPManagedSpace`, the memory migrates between the CPU and the GPU if:

- the hardware supports it and
- the kernel was compiled to support page migration and
- the environment variable `HSA_XNACK` is set to 1

L1 cache much smaller than on other GPUs. Decreasing memory usage via Enabling `Kokkos_ENABLE_HIP_MULTIPLE_KERNEL_INSTANTIATIONS` might improve performance but increases compilation time.

Using `-munsafe-fp-atomics` forces hardware-based floating-point atomics (no synchronization across kernels).

Supercomputer

- Aurora(ANL)

Unsupported features

- `WorkGraphPolicy`
- `Tasks`
- `Graphs` (dummy implementation)
- Virtual functions/function pointer
  `https://github.com/intel/llvm/pull/10540`
- `::printf` $\rightarrow$ `Kokkos::printf`

SYCL: CMake Configuration

```
cmake ..\
  -DCMAKE_CXX_COMPILER=icpx \
  -DKokkos_ENABLE_SYCL=ON \
  -DKokkos_ARCH_NATIVE=ON \
  -DKokkos_ARCH_INTEL_PVC=ON
```

- Replace last line with `-DKokkos_ARCH_INTEL_GEN=ON` for JIT compilation.
- Minimum version `oneAPI` 2023.0.0.

SYCL: Performance Intel Data Center GPU Max 1550, one tile

Results from `bytes_and_flops(TeamPolicy)`[3]

scalar	Bandwidth	Compute	Cache
`float`	1002 GiB/s	17484 GFlop/s	4973 GiB/s
`double`	960 GiB/s	8746 GFlop/s	6928 GiB/s
`int32_t`	1007 GiB/s	6108 GFlop/s	4714 GiB/s
`int64_t`	958 GiB/s	982 GFlop/s	4715 GiB/s

- Peak FP64 Vector: 22.9 TFLOPS/tile
- Memory Bandwidth: 1.6 TB/sec/tile
- Cache Size: 128KB (per work group)/408 MB

[3]This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

SYCL: Optimizing workgroup size

For `Kokkos::RangePolicy` with `Kokkos:parallel_for`, the workgroup size can be manually specified:

```
Kokkos::parallel_for(
  Kokkos::RangePolicy<ExecutionSpace>(space, 0, N)
    .set_chunk_size(1024),
  *this);
```

The subgroup size can be forced via

```
export IGC_ForceOCLSIMDWidth=32
```

or as compiler flag

```
-fsycl-default-sub-group-size=32
```

Also, see https://github.com/kokkos/kokkos/pull/6496, that would allow

```
Kokkos::RangePolicy<ExecSpace, Kokkos::SubGroupSize<16>>
```

SYCL: More optimizations I

Larger amount of registers can be requested via

```
export SYCL_PROGRAM_COMPILE_OPTIONS=\
  "-ze-opt-large-register-file"
```

or as compiler flag

```
-Xs "-options␣-ze-opt-large-register-file"
```

useful if there are spills

> *[...] compiled SIMD32 allocated 128 regs and spilled around 4*

SYCL: More optimizations II

Using https://github.com/kokkos/kokkos/pull/5879 allows specifying the scratch level at compile-time and helps with

> *[...] warning: Adding 4 occurrences of additional control flow due to presence of generic address space operations*

Multidimensional `View` access produces poor assembly. Replace

```
for (int i=0; i<N; ++i)
  view(i,3,10) = ...;
```

with

```
auto stride = view.stride(0);
auto view_ptr = &view(0,3,10);
for (int i=0; i<N; ++i, view_ptr+=stride)
  *view_ptr = ...;
```

Questions?