BENCHMARKING PORTABLE STAGGERED FERMION KERNEL WRITTEN IN KOKKOS AND MPI

LATTICE 2023

31st July, 2023 | Simon Schlepphorst | Jülich Supercomputing Centre
Staggered fermions
Quick recap

staggered fermionic action

\[ S_F[\chi, \bar{\chi}] = a^4 \sum_{n \in \Lambda} \bar{\chi}(n) \left( \sum_{\mu = 1}^{4} \eta_{\mu}(n) \frac{U_{\mu}(n)\chi(n + \hat{\mu}) - U_{\mu}^{\dagger}(n - \hat{\mu})\chi(n - \hat{\mu})}{2a} + m\chi(n) \right) \]

arithmetic intensity

\[ I = \frac{570 \text{ FLOP}}{792 \text{ B}} = 0.72 \text{ FLOP}/\text{B}. \]
Kokkos C++ Performance Portability EcoSystem

- writing modern C++ applications in a hardware agnostic way

---

Kokkos C++ Performance Portability EcoSystem

- writing modern C++ applications in a hardware agnostic way
- configured with CMake

---

Kokkos C++ Performance Portability EcoSystem

- writing modern C++ applications in a hardware agnostic way
- configured with CMake
- defaults to best memory layout for target architecture

Kokkos C++ Performance Portability EcoSystem

- writing modern C++ applications in a hardware agnostic way
- configured with CMake
- defaults to best memory layout for target architecture
  - LayoutLeft (column-major) for GPUs

---

Kokkos C++ Performance Portability EcoSystem

- writing modern C++ applications in a hardware agnostic way
- configured with CMake
- defaults to best memory layout for target architecture
  - LayoutLeft (column-major) for GPUs
  - LayoutRight (row-major) for CPUs

---

Kokkos C++ Performance Portability EcoSystem

- writing modern C++ applications in a hardware agnostic way
- configured with CMake
- defaults to best memory layout for target architecture
  - LayoutLeft (column-major) for GPUs
  - LayoutRight (row-major) for CPUs

C++ code

```cpp
using complex_t = Kokkos::complex<float>;
using Site = Kokkos::View<complex_t ****[3];
using Link = Kokkos::View<complex_t ****[4][3][3];
```

---

Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
- extend lattice by 2 in each direction to use for communication
Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
- extend lattice by 2 in each direction to use for communication
- only use this extension if MPI is used in direction of this hypercube face
Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
- extend lattice by 2 in each direction to use for communication
- only use this extension if MPI is used in direction of this hypercube face
  - split of this faces as halo and compute independly from the bulk
Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
- extend lattice by 2 in each direction to use for communication
- only use this extension if MPI is used in direction of this hypercube face
  - split of this faces as halo and compute independly from the bulk
  - use a continuous memory region as buffer for MPI
Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
- extend lattice by 2 in each direction to use for communication
- only use this extension if MPI is used in direction of this hypercube face
  - split of this faces as halo and compute independently from the bulk
  - use a continuous memory region as buffer for MPI
- otherwise apply periodic boundary conditions by recalculating the site indices
Message Passing Interface (MPI)

- communicate between processes to scale on massive parallel machines
- extend lattice by 2 in each direction to use for communication
- only use this extension if MPI is used in direction of this hypercube face
  - split of this faces as halo and compute independently from the bulk
  - use a continuous memory region as buffer for MPI
- otherwise apply periodic boundary conditions by recalculating the site indices

C++ code

```cpp
using BulkSpace_t = Kokkos::DefaultExecutionSpace;
using HaloSpace_t = Kokkos::DefaultExecutionSpace;

BulkSpace_t BulkExecSpace = BulkSpace_t();
HaloSpace_t HaloExecSpcae = HaloSpace_t();
Kokkos::fence(); // barrier for all execution spaces
HaloExecSpcae.fence(); // barrier for only one execution space
```
### Kernel Algorithm

**Kernel (Input: \( U_\mu, \chi_{in} \) Output: \( \chi_{out} \)**)

<table>
<thead>
<tr>
<th>( n \in \Lambda )</th>
</tr>
</thead>
<tbody>
<tr>
<td>for ( i \leftarrow 1, 2, 3 ) do</td>
</tr>
<tr>
<td>( t \leftarrow 0 )</td>
</tr>
<tr>
<td>for ( j \leftarrow 1, 2, 3 ) do</td>
</tr>
<tr>
<td>for ( \mu \leftarrow 1, 2, 3, 4 ) do</td>
</tr>
<tr>
<td>( t \leftarrow t + U_\mu(n)<em>{ij} \cdot \chi</em>{in}(p(n + \hat{\mu}))_j )</td>
</tr>
<tr>
<td>( t \leftarrow t - U_\mu(p(n - \hat{\mu}))<em>{ji} \cdot \chi</em>{in}(p(n - \hat{\mu}))_j )</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>( \chi_{out,i} \leftarrow t )</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

- \( p() \) calculates the correct \( n \) according to periodic boundaries
AMD Ryzen 7742 (x86 CPU, Dual Socket)

![Graph showing bandwidth vs. time for different compilers (AOCC, Clang, GCC) on a JURECA DC system with Kokkos 3.6, AOCC 3.2, Clang 13.0, GCC 11.2).]

**JURECA DC @ JSC, Kokkos 3.6, AOCC 3.2, Clang 13.0, GCC 11.2**
Fujitsu A64FX (ARM CPU)

CTE-ARM @ BSC, Kokkos 3.6, GCC 11.1, Clang 14.0
Nvidia A100 (GPU)

JURECA DC @ JSC, Kokkos 3.6, GCC 11.2, NVHPC 22.1, CUDA 11.5
Nvidia A100 (GPU) - Full node

Bandwidth [TB/s] vs. L/T

- 1 GPU
- 2 GPUs
- 4 GPUs

JURECA DC @ JSC, Kokkos 3.6, GCC 11.2, NVHPC 22.1, CUDA 11.5, PSMPI 5.5.0
AMD MI250 (GPU) - one Graphics Compute Die (GCD)

Peak HBM Bandwidth per GCD

JURECA DC Evaluation Platform @ JSC, Kokkos 3.6, Clang 14.0, ROCm 5.2
AMD MI250 (GPU) - Full node

JURECA DC Evaluation Platform @ JSC, Kokkos 3.6, Clang 14.0, ROCm 5.2, OpenMPI 4.1.2
Nvidia A100 vs. AMD MI250 (GPU)

Number of NVIDIA GPUs vs. Bandwidth (TB/s)
Nvidia H100 PCIe (GPU)

Bandwidth [TB/s]

Peak HBM Bandwidth

JURECA DC Evaluation Platform @ JSC, Kokkos 4.0, GCC 11.3, CUDA 12.0, LaunchBounds (384,1)
Nvidia H100 PCIe vs. Nvidia A100 (GPU)

JURECA DC Evaluation Platform @ JSC, Kokkos 4.0, GCC 11.3, CUDA 12.0, LaunchBounds (384,1)
Thank you for your attention!