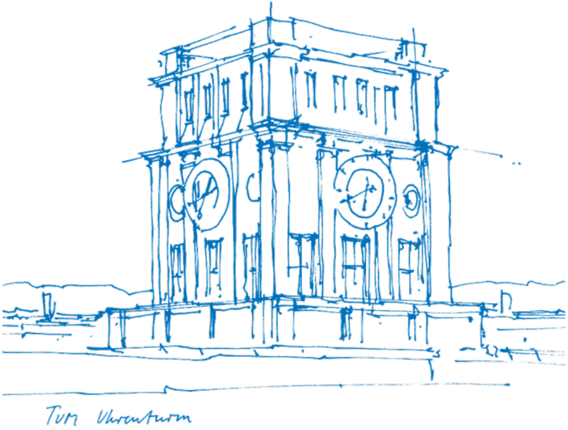
Chair of Scientific Computing
TUM School of Computation, Information and Technology
Technical University of Munich

An Introduction to Performing Parallel Programming with Kokkos



Yuan Luo

Chair of Scientific Computing
TUM School of Computation, Information and Technology
Technical University of Munich

July 4th, 2023

Research Background

There are so many parallel programming models targeting different computing devices.

(a) CUDA[1] **(b)** OpenMP[2] **(c)** ROCm[3]

Each of them has its own unique semantic rules, which makes it harder to switch from one model to another one.

[1] CUDA: `https://developer.nvidia.com/cuda-toolkit`
[2] OpenMP: `https://www.openmp.org/`
[3] ROCm: `https://www.amd.com/en/graphics/servers-solutions-rocm`

Research Background

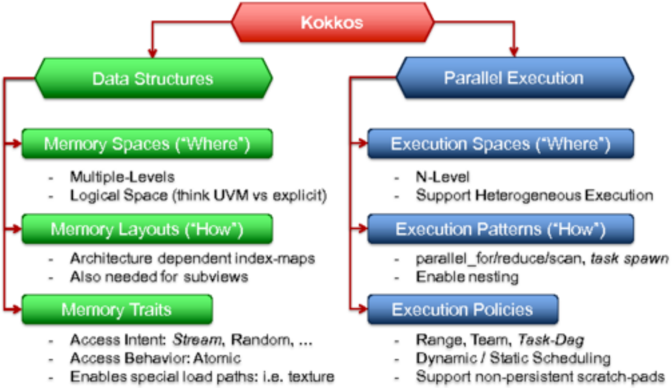
Kokkos: https://kokkos.org/

Kokkos is a programming model in C++ for writing performance portable applications targeting all major HPC platforms.

It has been used for many scientific applications, such as **molecular dynamics simulation**, **quantum chromodynamics**, and **astrophysics simulations**.

A comprehensive study on Kokkos should be carried out to see the tradeoff Kokkos offers.

Kokkos’ Core Abstractions



Kokkos

Data Structures

Memory Spaces (“Where”)

- Multiple-Levels
- Logical Space (think UVM vs explicit)

Memory Layouts (“How”)

- Architecture dependent index-maps
- Also needed for subviews

Memory Traits

- Access Intent: *Stream*, Random, …
- Access Behavior: Atomic
- Enables special load paths: i.e. texture

Parallel Execution

Execution Spaces (“Where”)

- N-Level
- Support Heterogeneous Execution

Execution Patterns (“How”)

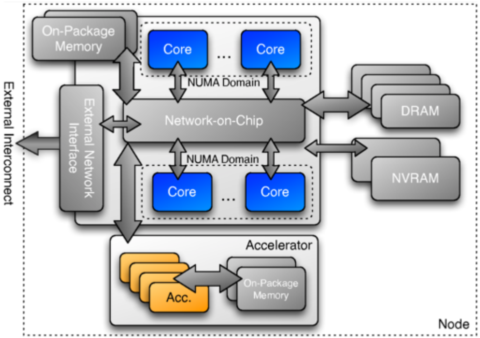
- parallel_for/reduce/scan, *task spawn*
- Enable nesting

Execution Policies

- Range, Team, *Task-Dag*
- Dynamic / Static Scheduling
- Support non-persistent scratch-pads

https://kokkos.github.io/kokkos-core-wiki/

Execution Spaces



https://kokkos.github.io/kokkos-core-wiki/

In the computing node of an HPC, there might be multiple computing resources: CPU, GPU, FPGA etc.

Execution Spaces

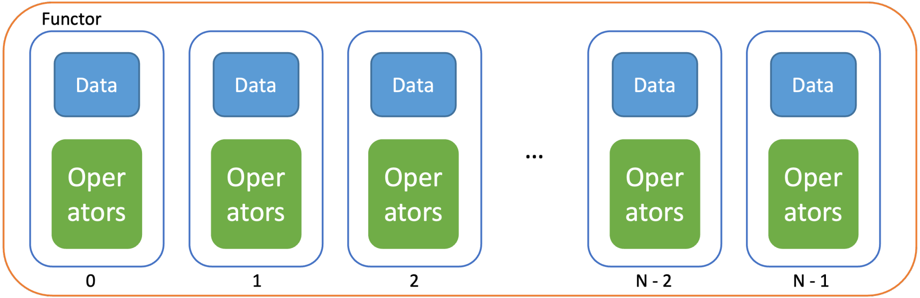
Kokkos has categorized all the possible execution spaces into three classes:

- Host Serial: A serial execution on the CPU (**Kokkos::Serial**)
- Host Parallel: OpenMP and Pthreads (**Kokkos::OpenMP**)
- Device Parallel: CUDA, HIP and OpenMPTarget (**Kokkos::Cuda**, **Kokkos:OpenMPTarget**)

Functor

Pack **data** and **operators** together.

Parallel_for(N, functor)



Execution Pattern

- **Parallel_for ≈ #pragma omp parallel for**
- **Parallel_reduce ≈ #pragma omp parallel for reduce**: reduce elements based on the given operator
- **Parallel_scan ≈ MPI_scan()**: Reduction using a prefix operation

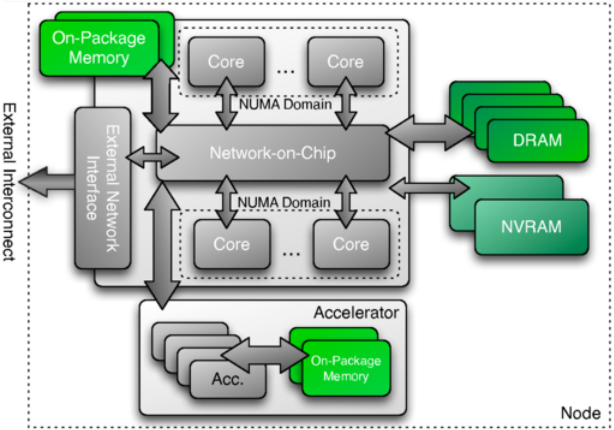
```
View<double**> A("a", M, N);
// ... fill A with some numbers ...
double result = 0.0;
Kokkos::parallel_reduce( N, KOKKOS_LAMBDA ( int j, double &sum ) {
    for ( int i = 0; i < M; i++)  sum += A(j, i);
}, result);
```

Execution Policies

- **Range Policy**: the simplest form
- **MDRange Policy**: for a multiple dimensional iteration space
- **Team Policy**: organize threads into thread teams (Think it as CUDA thread block)

```
Kokkos::parallel_for(Kokkos::MDRangePolicy<Kokkos::Rank<2>>({0, 0}, {N,
    N}), KOKKOS_LAMBDA(const int i, const int j) {
        printf("MDRange Policy: i = %d, j = %d\n", i, j);
    });
});
```

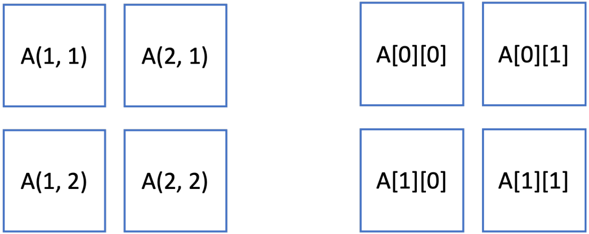
Memory Spaces



https://kokkos.github.io/kokkos-core-wiki/

Memory Layouts

Arrays in FORTRAN are stored in **column-major** order, while arrays in C are stored in **row-major** order. The language-prescribed memory layout limits the portability of a program.



A(1, 1)	A(2, 1)
A(1, 2)	A(2, 2)

Fortran

A[0][0]	A[0][1]
A[1][0]	A[1][1]

C

In Kokkos, a data structure can be instantiated with different layouts at compile time or runtime, allowing for **architecture-independent** optimizations.

Memory Traits

Memory Traits in Kokkos specify how a data structure is accessed in an algorithm.

It serves as hints to the compiler or runtime system about the intended access patterns of the data.

For example:

- **Kokkos::RandomAccess**: optimize the memory access pattern for efficient random access operations.
- **Kokkos::Atomic**: apply the necessary synchronization mechanisms to ensure correct atomic updates.

View

A View in Kokkos is a lightweight C++ class that represents an array or a multidimensional data structure.

```
View<double*, Device> oneDimensionArray("one", 10);
View<double**, Device> twoDimensionArray("two", 5, 5);

// memory spaces
Kokkos::View<double*, Kokkos::HostSpace> hostView("host_view", N);
Kokkos::View<double*, Kokkos::CudaSpace> deviceView("device_view", N);

// memory traits
Kokkos::View<int*, Kokkos::Atomic> data("data", N);
```

Testing Platform and Benchmark

Testing Platform

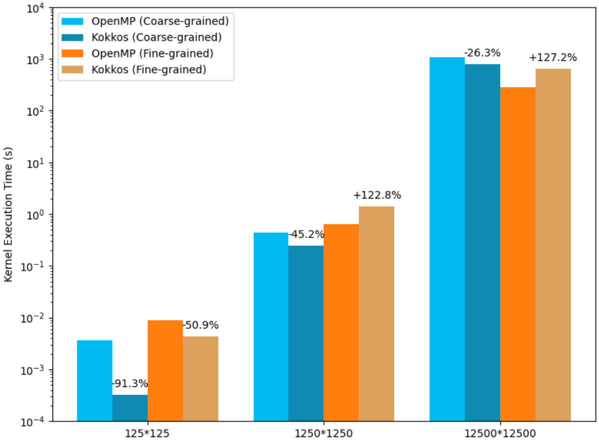
- Intel Xeon Processor E5-2620 v4 CPU x 2 (32 Cores in total)
- 128 GB RAM
- Nvidia Tesla P100 x 1
- Kokkos(4.0.0), OpenMP(4.1.5), and CUDA(11.2)

Benchmark: Matrix Multiplication: coarse- and fine-grained

To make the comparison as fair as possible:

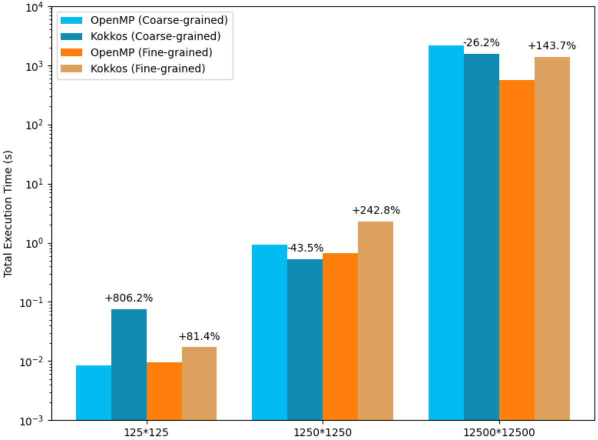
- The same Kokkos program will be compiled into both OpenMP version and CUDA version.
- To what extent should I optimize the programs?

Performance Comparison with OpenMP



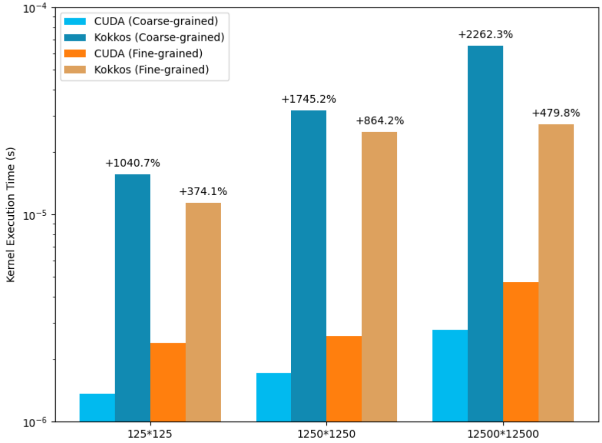
Kernel Execution Time

Performance Comparison with OpenMP



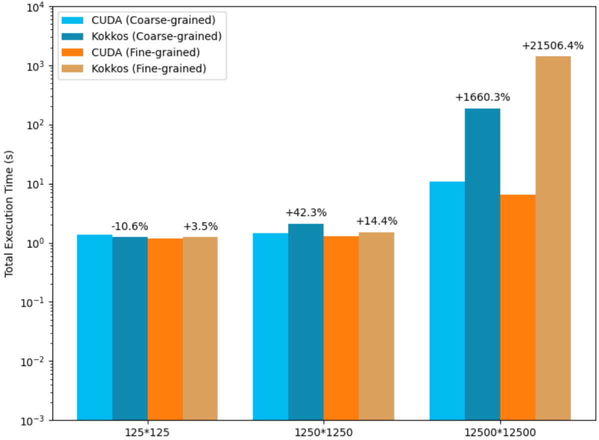
Total Execution Time

Performance Comparison with CUDA



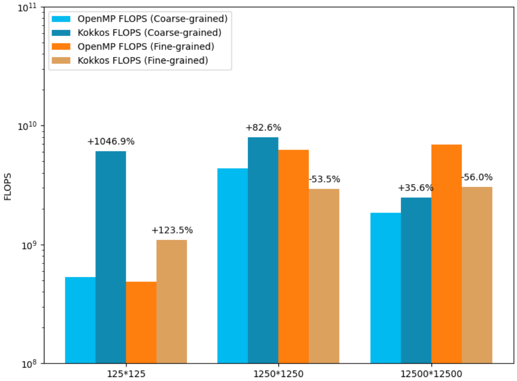
Kernel Execution Time

Performance Comparison with CUDA

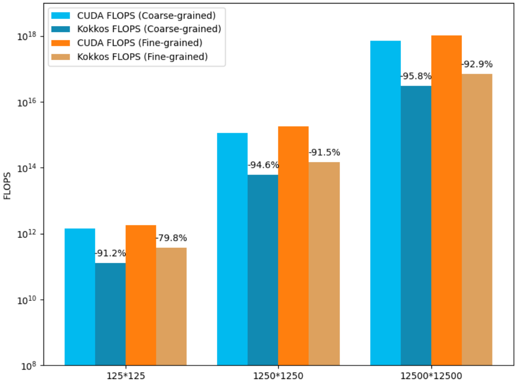


Total Execution Time

Performance Comparison of Kokkos Program on Different Backends



(a) OpenMP FLOPS



(b) CUDA FLOPS

Energy Consumption

Due to the limitation of the testing platform, no data on energy consumption of the CPU can be retrieved.

Kokkos CUDA program consumes **more energy** on the GPU due to the fact that it takes **longer** to finalize the program.

The power usage of both programs peaks at roughly **75** watts. After the intensive calculation phase, the power consumption of both programs is roughly **35** watt.

Code Readability and Portability

They are both subjective matters:

- Code Readability: OpenMP $>$ Kokkos $>$ CUDA
- Portability: Kokkos $>$ OpenMP $>$ CUDA

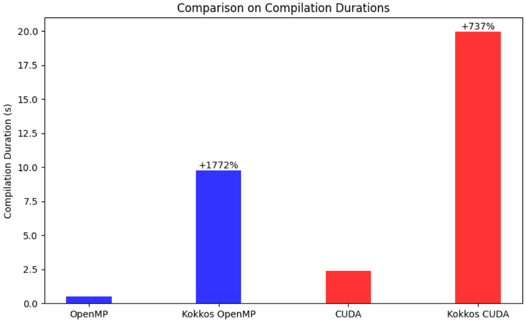
Community and Support

OpenMP has an active community with a dedicated web- site and forum for discussions and information sharing (www.openmp.org).

NVIDIA’s **CUDA** is supported via the NVIDIA Developer Program, which provides information and a developer forum (developer.nvidia.com).

Kokkos has a burgeoning community focused on its GitHub repository (github.com/kokkos/kokkos), which offers access to source code, examples, and collaborative development initiatives.

Compilation Duration



For this small project, Kokkos has an overall higher compilation duration.

However, for larger projects, this might not be a problem. Kokkos has leveraged techniques like caching mechanisms to speed up the compilation.

Future Work

- Study the finalization process for Kokkos CUDA: what takes it so long
- Test the compilation duration with larger projects
- Conduct more precise energy consumption tests
- Try to find deeper explanations for some cases showed in the paper

Summary

- The core abstractions in Kokkos
- OpenMP vs. Kokkos
- CUDA vs. Kokkos
- Kokkos’ performance portability