kokkos

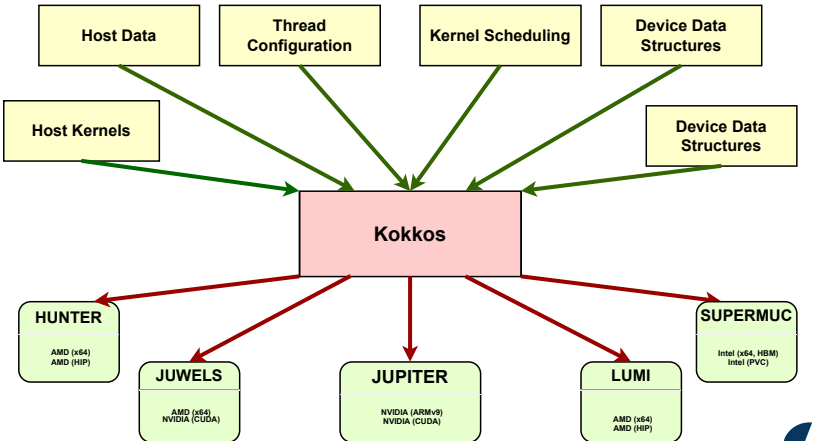
A FIRST INTRODUCTION

NatESM Workshop

November 6, 2024 | Dr. Jayesh Badwaik | Jülich Supercomputing Centre

PERFORMANCE PORTABILITY

Performant Single Source Implementation



JÜLICH Forschungszentrum

HISTORY AND SUPPORT

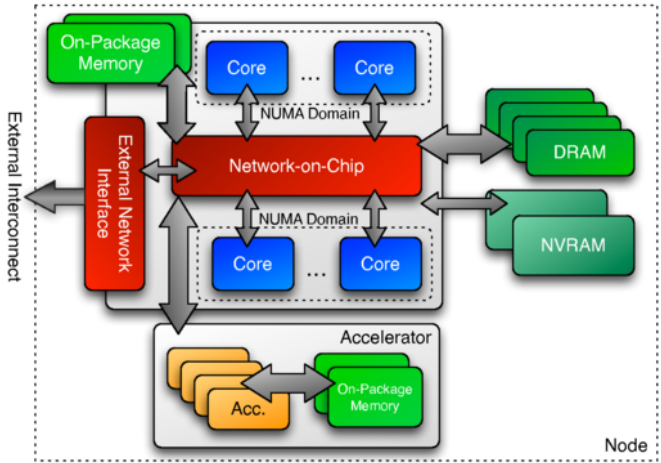
- Established 2012
- Widely used in HPC
- Support for most major HPC platforms

- Feedback loop with C++ Standards
  - Parallel STL
  - `std::atomic_ref`
  - `std::mdspan` and `std::mdarray`

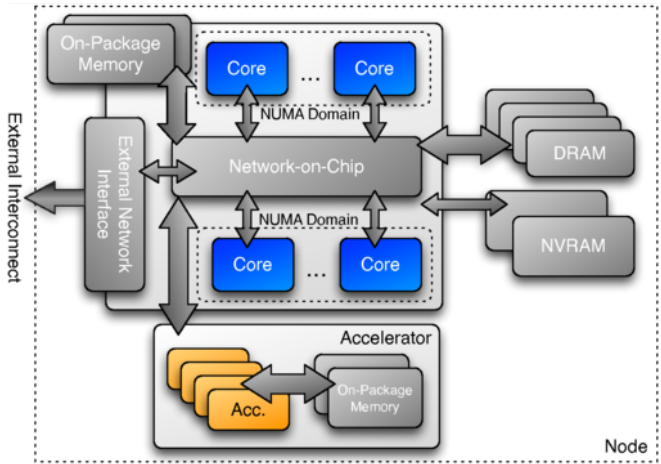
Online Presence

- `https://github.com/kokkos`
  - Primary Github Organization
- `https://kokkosteam.slack.com`
  - Slack Channel for Kokkos

MACHINE MODEL

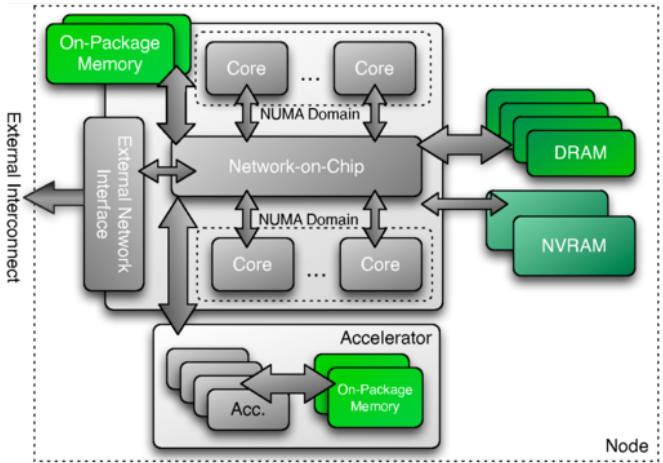


EXECUTION MODEL – EXECUTIONPOLICY



JÜLICH Forschungszentrum

MEMORY MODEL – MEMORYSPACE



JÜLICH Forschungszentrum

FIRST LOOK

Setup your Environment

```
$ cd kokkos/tasks
$ source setup.sh
$ make run0
# ... some slurm output
# ... some program output
```

OUTPUT

```
Hello from i = 0
Hello from i = 1
Hello from i = 2
Hello from i = 3
Hello from i = 4
Hello from i = 5
Hello from i = 6
Hello from i = 7
```

HELLO, WORLD!

```
struct hello_type {
  __host__ __device__ void operator()(const int i) const {
    Kokkos::printf("Hello from i = %i\n", i);
}};

int main(int argc, char* argv[]) {
  Kokkos::initialize(argc, argv);
  auto hello = hello_type();
  Kokkos::parallel_for("HW", 8, hello);
  Kokkos::finalize();
}
```

FIRST LOOK

Hello, World!

- The Call Invocation

```
Kokkos::parallel_for("HelloWorld", 15, hello);
```

Pattern: `parallel_for`, Policy: 15 threads Body: `hello()`

- The Body (Functor of Function Object)

```
struct hello_type {
  __host__ __device__
  void operator()(const int i) const {
    Kokkos::printf("Hello from i = %i\n", i);
  }
};
```

FIRST LOOK

```
Kokkos::parallel_for("HW", 15, hello);
```

FIRST LOOK

```
Kokkos::parallel_for("HW", 15, hello);
Kokkos::parallel_for("HW", RangePolicy<>(15), hello);
```

FIRST LOOK

```
Kokkos::parallel_for("HW", 15, hello);
Kokkos::parallel_for("HW", RangePolicy<>(15), hello);
Kokkos::parallel_for("HW", RangePolicy<DefaultExecutionSpace>(15), hello);
```

FIRST LOOK

```
Kokkos::parallel_for("HW", 15, hello);
Kokkos::parallel_for("HW", RangePolicy<>(15), hello);
Kokkos::parallel_for("HW", RangePolicy<DefaultExecutionSpace>(15), hello);
template<class ExecPol, class FType>
parallel_for(const std::string &name, const ExecPol &policy, const FType &functor);
```

EXECUTION SPACES

Execution Policies: Control how the code runs

Policy	Description
`IntegerType`	`RangePolicy(0, n)`
`RangePolicy`	One-dimensional range
`MDRangePolicy`	Multidimensional range

Execution Spaces: Control where the code runs

- CPU Serial Backend: Serial
- CPU Parallel Backends: OpenMP, Threads
- Device Backends: CUDA, HIP, ...

Important Execution Spaces

- `DefaultExecutionSpace`
  `device` > `host-parallel` > `host-serial`
- `DefaultHostExecutionSpace`
  `host-parallel` > `host-serial`

LAMBDAS (CLOSURES)

```
[captures] (params) -> return-type {
  <function-body>;
}
```

LAMBDAS (CLOSURES)

```
  [captures] (params) -> return-type {
    <function-body>;
  }

[=] __host__ __device__ (const int i)
{
  Kokkos::printf("Hello from i = %i\n", i);
}
```

LAMBDAS (CLOSURES)

```
[captures] (params) -> return-type {
  <function-body>;
}
```

```
[=] __host__ __device__ (const int i)
{
  Kokkos::printf("Hello from i = %i\n", i);
}
```

```
struct hello_type {
  __host__ __device__
  void operator()(const int i) const {
    Kokkos::printf("Hello from i = %i\n", i);
  }
};
```

LAMBDAS (CLOSURES)

```
[captures] (params) -> return-type {
  <function-body>;
}
```

Task 02: Rewrite the `hello_type` function object as a lambda

LAMBDAS (CLOSURES)

```
[=] __host__ __device__ (const int i)
{
  Kokkos::printf("Hello from i = %i\n", i);
}
```

```
struct hello_type {
  __host__ __device__
  void operator()(const int i) const {
    Kokkos::printf("Hello from i = %i\n", i);
  }
};
```

LAMBDAS (CLOSURES)

```
[=] __host__ __device__ (const int i)
{
  Kokkos::printf("Hello from i = %i\n", i);
}
```

```
struct hello_type {
  __host__ __device__
  void operator()(const int i) const {
    Kokkos::printf("Hello from i = %i\n", i);
  }
};
```

```
[=] KOKKOS_INLINE_FUNCTION (const int i)
{
  Kokkos::printf("Hello from i = %i\n", i);
}
```

```
struct hello_type {
  KOKKOS_INLINE_FUNCTION
  void operator()(const int i) const {
    Kokkos::printf("Hello from i = %i\n", i);
  }
};
```

SIMPLE REDUCE

The Problem

$$S = \sum_{i=1}^{n} i^2 \tag{1}$$

OpenMP Version

```
std::atomic_int sum(0);
#pragma omp parallel for
for (int i=0; i<n; i++){
  sum += i * i;
}
```

STL Serial Version

```
auto view = std::ranges::views::iota(0, n);
std::reduce(std::begin(view), std::end(view), 0,
  [](int a, int b){ return a + b*b; });
```

SIMPLE REDUCE

Kokkos Version

- The Function Object

```
struct squaresum {
  using value_type = int;
  KOKKOS_INLINE_FUNCTION
  void operator()(const int i, int& lsum) const {
    lsum += i * i;
  }
};
```

- `Kokkos::parallel_reduce(n, squaresum(), sum);`
- `lsum` : **thread-local** variable used internally by Kokkos
- **Task 03** : Write a parallel reduce operation to calculate the sum of squares

SUMMARY

- Pattern, Policy, Body Paradigm
- `Kokkos::parallel_for`
- Execution Spaces
- `Kokkos::parallel_reduce`

VIEWS

- Datatype for multidimensional array
- Reference Semantics
- No allocations unless explicitly specified
- Automatic deallocation by reference counting
- Rank fixed at compile time
- Rectangular arrays only
- Dimension size: compile-time/runtime
- Access via `(...)` operator

```
View < double *** > data ("label" , N0 , N1 , N2 ); //3 runtime, 0 compile
View < double **[ N2 ] > data ("label" , N0 , N1 ); //2 runtime, 1 compile
View < double *[ N1 ][ N2 ] > data ("label" , N0 ); //1 runtime, 2 compile
View < double [ N0 ][ N1 ][ N2 ] > data ("label" ); //0 runtime, 3 compile
```

```
data(i,j,k) = 0.0;
```

```
std::cout<<data(i,j,k);
```

REFERENCE SEMANTICS

- Allocations only happen when explicitly specified – **no hidden allocations**
- Copy and assignment are **shallow**

```
View < double *[5] > a ( "a" , N );
a (0 ,2) = 1; // a(0,2);

View < double *[5] > b ( "b" , K );
a = b;
b (0 ,2) = 2; // a(0,2) = b(0,2) = c(0,2) = 2

View < double ** > c ( b );
c (0 ,2) = 3; // a(0,2) = b(0,2) = c(0,2) = 3
```

SIMPLE VIEW EXAMPLE

```
using view_type = Kokkos::View<double*[3]>;

struct InitView {
  view_type a;

  InitView(view_type a_) : a(a_) {}

  KOKKOS_INLINE_FUNCTION
  void operator()(const int i) const {
    a(i, 1) = 1.0 * i * i;
    a(i, 2) = 1.0 * i * i * i;
}};

b = view_type();
// initialize b
Kokkos::parallel_for(N, InitView(b));
```

MEMORY SPACES AND EXECUTION SPACES

```
using view_type = Kokkos::View<double*[3]>;
Kokkos::parallel_for(N, InitView(a));
```

MEMORY SPACES AND EXECUTION SPACES

```
using view_type = Kokkos::View<double*[3]>;
Kokkos::parallel_for(N, InitView(a));
```

```
Kokkos::View<double * [3], DefaultMemorySpace> a("A", N);
Kokkos::parallel_for(RangePolicy<DefaultExecutionSpace>(0,N), InitView(a));
```

MEMORY SPACES AND EXECUTION SPACES

```
using view_type = Kokkos::View<double*[3]>;
Kokkos::parallel_for(N, InitView(a));
Kokkos::View<double * [3], DefaultMemorySpace> a("A", N);
Kokkos::parallel_for(RangePolicy<DefaultExecutionSpace>(0,N), InitView(a));
```

- Available memory spaces: `HostSpace`, `CudaSpace`, `CudaUVMSpace`
- Default memory space associated with the execution space

```
View < double* > a ( "A" ,N ); // Equivalent
View < double*, DefaultExecutionSpace::memory_space>b ( "B" ,N ); // Equivalent
```

MEMORY SPACES AND EXECUTION SPACES

```
using view_type = Kokkos::View<double*[3]>;
Kokkos::parallel_for(N, InitView(a));
```

```
Kokkos::View<double * [3], DefaultMemorySpace> a("A", N);
Kokkos::parallel_for(RangePolicy<DefaultExecutionSpace>(0,N), InitView(a));
```

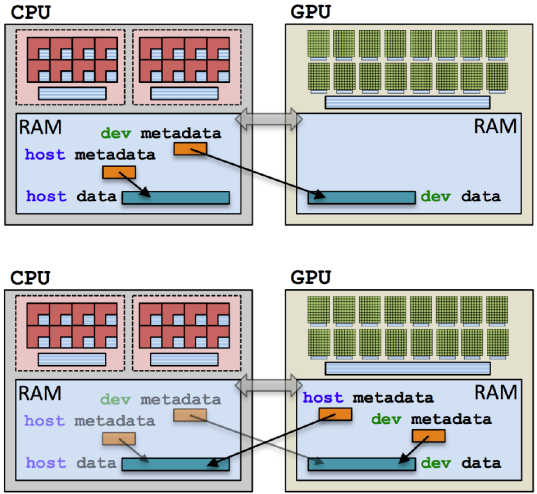
- Available memory spaces: `HostSpace`, `CudaSpace`, `CudaUVMSpace`
- Default memory space associated with the execution space

```
View < double* > a ( "A" ,N ); // Equivalent
View < double*, DefaultExecutionSpace::memory_space>b ( "B" ,N ); // Equivalent
```

Important Memory Spaces

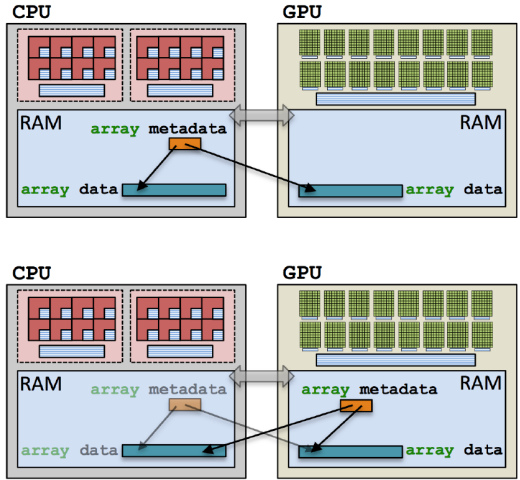
- `HostSpace`
- `SharedSpace`
- `SharedHostPinnedSpace`

HOST AND DEVICE SPACE



JÜLICH Forschungszentrum

UVM SPACE

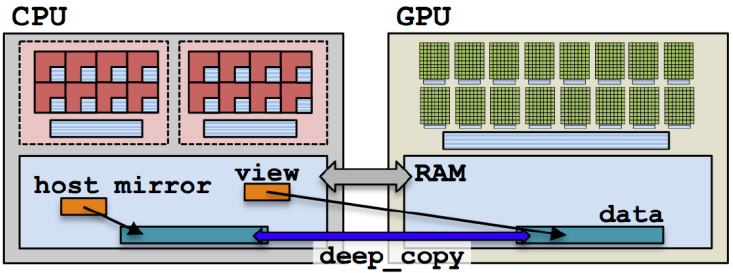


JÜLICH Forschungszentrum

DEEP COPY WITH MIRRORING

Mirroring

```
using view_type = Kokkos::View<double**, Space>;
using host_view_Type = view_type::HostMirror;
view_type view (...);
host_view_type copy_view = Kokkos::create_mirror_view(view);
Kokkos::deep_copy(copy_view, view);
```



JÜLICH Forschungszentrum

DEEP COPY WITH MIRRORING

Mirroring

Task 4: Deep copy the view a to b and print the values of b

MULTIDIMENSIONAL LOOPS

Consider nested loops:

```
for(int i = 0; i < N0; ++i) {
  for (int j = 0; j < N1 ; ++ j ) {
    for (int k = 0; k < N2 ; ++ k ) {
     some_function (i,j,k);
    }
  }
}
```

Single Dimensional Loop

```
Kokkos::parallel_for("mdloop", N0,
  KOKKOS_LAMBDA(const i) {
  for(int j = 0; j < N1 ; ++j ) {
    for(int k = 0; k < N2 ; ++k ) {
      some_function (i , j , k );
    }
  }
});
```

- Parallelization only of outer loop
- Only $N0 \times N1 \times N2$ iterations might be worth parallelizing

MULTIDIMENSIONAL LOOPS

MDRangePolicy

Parallel For

```
parallel_for("mdloop", Kokkos::MDRangePolicy<Rank<3>>({0, 0, 0} ,{ N0, N1, N2}),
KOKKOS_LAMBDA ( int64_t i , int64_t j , int64_t k ) {
  some_function(i, j, k);
});
```

Parallel Reduce

```
parallel_reduce("mdloop", MDRangePolicy<Rank<3>>({0, 0, 0} ,{ N0, N1, N2}),
KOKKOS_LAMBDA ( int64_t i , int64_t j , int64_t k, double& lsum) {
  lsum += some_function(i, j, k);
});
```

HANDS-ON EXERCISE

RangePolicy vs MDRangePolicy

task05

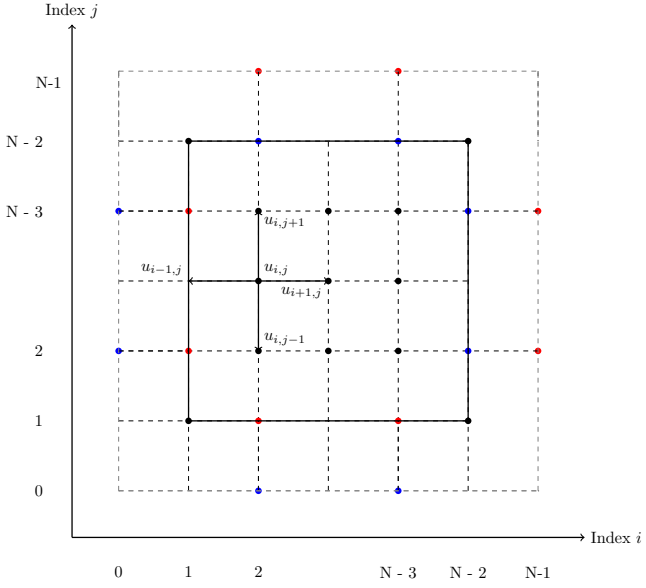
SUMMARY AND ADVANCED TOPICS

- RangePolicy – 1D
- MDRangePolicy – Multi-Dimensional
- TeamPolicy – Hirearchial Parallelism
- Views – Multidimensional Arrays
- Mirror Views – Shallow and Deep Copies.
- Unified Virtual Memory – Managed Data Copy
- Subview – Slice, Strided View
- Layouts – Row-Major, Column-Major, Strides
- Unmanaged Views – Wrap existing allocations
- Dual Views - Transition from CPU to GPU

JACOBI SOLVER FOR POISSON EQUATION

$$\Delta u = f$$

$$u_0(x, y) = 0 \qquad f = e^{-10(x^2+y^2)}$$


Index j
N-1
N - 2
N - 3
2
1
0
$u_{i,j+1}$
$u_{i-1,j}$
$u_{i,j}$
$u_{i+1,j}$
$u_{i,j-1}$
Index i
0
1
2
N - 3
N - 2
N-1

JACOBI SOLVER FOR POISSON EQUATION

$$u_{i,j}^{n+1} = \frac{1}{4}\left[u_{i-1,j}^{n} + u_{i+1,j}^{n} + u_{i,j-1}^{n} + u_{i,j+1}^{n} - f_{i,j}\right]$$

$$u_{0,i}^{n+1} = u_{N-2,i}^{n} \qquad u_{0,j}^{n+1} = u_{j,N-2}^{n}$$

$$u_{N-1,i}^{n+1} = u_{0,i}^{n} \qquad u_{N-1,j}^{n+1} = u_{j,0}^{n}$$