OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

March 19 | Kokkos Tea-Time

Kokkos Best Practices – Integrating Kokkos into Your Project

Damien Lebrun-Grandié

Computational Sciences and Engineering Division

U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY

Getting started with Kokkos

Part 1:
Integrate Kokkos into your CMake project

Part 2:
Initialize the Kokkos execution environment and start using it

kokkos

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Part 1:
Integrate Kokkos into your CMake project

Building Kokkos
A little historical context

The Dark Ages (Kokkos 2.X)

Generated GNU Makefiles

Users have to write their own FindKokkos.cmake module

GNU Make

The Middle Ages (Kokkos 3.X)

CMake

TriBITS

Generated GNU Makefiles

Today (Kokkos 4.X)

CMake

find_package()
add_subdirectory()

TriBITS gone

GNU Makefiles deprecated

How to integrate Kokkos into your CMake project (1/3)
External Kokkos (Recommended for most users)

Kokkos provides a Kokkos::kokkos target.

It automatically handles

- Include directories
- Link libraries
- Compiler options
- Etc.

```
find_package(Kokkos 4.2 REQUIRED CONFIG)
target_link_libraries(MyTarget PRIVATE Kokkos::kokkos)
```

```
> cmake -DKokkos_ROOT=/path/to/kokkos -B builddir
```

How to integrate Kokkos into your CMake project (2/3) Embedded Kokkos

via add_subdirectory() and Git Submodules

```
> git submodule add -b 4.5.01 \
  https://github.com/kokkos/kokkos.git \
  tpls/kokkos
> git commit -m 'Adding Kokkos v4.5.1 as a submodule'
```

```
add_subdirectory(tpls/kokkos)
target_link_libraries(MyTarget PRIVATE Kokkos::kokkos)
```

via FetchContent

```
include(FetchContent)
FetchContent_Declare(Kokkos
  URL
https://github.com/kokkos/kokkos/releases/download/4.5.01/kokkos-4.5.01.tar.gz
  URL_HASH
SHA256=52d003ffbbe05f30c89966e4009c017efb1662b02b2b73190670d3418719564c
)
FetchContent_MakeAvailable(Kokkos)
target_link_libraries(MyTarget PRIVATE Kokkos::kokkos)
```

How to integrate Kokkos into your CMake project (3/3) Supporting Both External and Embedded Kokkos

Control with:

CMAKE_DISABLE_FIND_PACKAGE_Kokkos:
Set to TRUE to force the use of the embedded Kokkos, even if an external installation is found.

CMAKE_REQUIRE_FIND_PACKAGE_Kokkos:
Set to TRUE to require an external Kokkos installation. The build will fail if Kokkos is not found.

Kokkos_ROOT:
Specify the directory where CMake should search for Kokkos when using find_package().

```
find_package(Kokkos CONFIG)
if(Kokkos_FOUND)
  message(STATUS
    "Found Kokkos: ${Kokkos_DIR} (version \"${Kokkos_VERSION}\")")
else()
  message(STATUS
    "Kokkos not found externally. Fetching via FetchContent.")
  include(FetchContent)
  FetchContent_Declare(Kokkos URL
    https://github.com/kokkos/kokkos/archive/refs/tags/4.4.01.tar.gz
  )
  FetchContent_MakeAvailable(Kokkos)
endif()
target_link_libraries(MyTarget PRIVATE Kokkos::kokkos)
```

Same as previous slide

Part 2:
Initialize the Kokkos execution environment and start using it

Include Kokkos Initialize/finalize

1. Include the header file.
2. Initialize Kokkos before you use it.
3. Finalize after you are done.

Similar to what you do with MPI

```
#include <mpi.h>
MPI_Init
MPI_Finalize
```

```
#include <Kokkos_Core.hpp>

int main(int argc, char** argv) {
  Kokkos::initialize(argc, argv);
  {
    Kokkos::View<int*> v("v", 5);
    Kokkos::parallel_for("fill", 5,
      KOKKOS_LAMBDA(int i) { v(i) = i; });
    int r;
    Kokkos::parallel_reduce("accumulate", 5,
      KOKKOS_LAMBDA(int i, int& partial_r) {
        partial_r += v(i);
      }, r);
    KOKKOS_ASSERT(r == 10);
  }
  Kokkos::finalize();

  return 0;
}
```

Using Kokkos with MPI

Initialize **MPI first** and **then Kokkos**.

Kokkos intelligently detects the MPI environment and automatically maps processes to available local GPUs using a round-robin method.

You can change the strategy by passing a command-line `--kokkos-map-device-id-by "random"` or by setting an environment variable `KOKKOS_MAP_DEVICE_ID_BY="random"`

Or you can force the GPU selection via `--kokkos-device-id=2` or `KOKKOS_DEVICE_ID=2`

```
int main(int argc, char** argv) {
  MPI_Init(&argc, &argv);
  Kokkos::initialize(argc, argv);
  {
    // ...
  }
  Kokkos::finalize();
  MPI_Finalize();

  return 0;
}
```

Programmatically passing initialization settings

Use `InitializationSettings` to define the settings for initializing Kokkos programmatically without having to call the two-parameter form (`argc` and `argv`)

Each setting has a corresponding

- `set_*`
- `has_*`
- `get_*`

member functions.

```
if (!settings.has_device_id())
  settings.set_device_id(1);

printf("Device ID: %d\n",
       settings.get_device_id());
```

```
int main() {
  Kokkos::initialize(
    Kokkos::InitializationSettings()
      .set_disable_warnings(true)
      .set_map_device_id_by("random")
      .set_num_threads(1)
  );

  // ...

  Kokkos::finalize();
}
```

RAII-style execution environment scope guards

RAII = Resource Acquisition Is Initialization

`ScopeGuard` calls `Kokkos::initialize` with the provided arguments in the constructor and `Kokkos::finalize` in the destructor.

```
#include <Kokkos_Core.hpp>
int main(int argc, char** argv) {

  Kokkos::ScopeGuard exec_env(argc, argv);

  Kokkos::View<int*> v("v", 5);
  Kokkos::parallel_for("fill", 5,
    KOKKOS_LAMBDA(int i) { v(i) = i; });
  int r;
  Kokkos::parallel_reduce("accumulate", 5,
    KOKKOS_LAMBDA(int i, int& partial_r) {
      partial_r += v(i);
    }, r);
  KOKKOS_ASSERT(r == 10);

  return 0;
} // v.~View() is called before finalize()
  // exec_env.~ScopeGuard() called
```

What if you don’t get to write main()

Initialize if not already initialized (LAST RESORT)

When users of your library are not aware that it uses Kokkos underneath the hood.

Detect if Kokkos was initialized and initialize if you must. You are then responsible for finalizing.

```
void initialize_if_necessary() {
  static bool once = [] {
    assert(!Kokkos::is_finalized());
    if (!Kokkos::is_initialized()) {
      Kokkos::initialize();
      std::atexit(Kokkos::finalize);
    }
    return true;
  }();
  assert(Kokkos::is_initialized());
}
```

Acquire/release resources

When Kokkos being initialized is a precondition for using your library; but you need to manage some resources that potentially persist across successive API calls.

```
void ensure_is_initialized() {
  static bool once = [] {
    assert(Kokkos::is_initialized());
    // acquire resources
    Kokkos::push_finalize_hook([]() {
      // release resources
    });
    return true;
  }();
  assert(!Kokkos::is_finalized());
}
```

The End

Funding Acknowledgements:
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.


Thank you for your attention.
Find us on https://kokkos.org