Compiling Kokkos for GPUs

Lars Pastewka

Table of contents

Overview . . . 1
Hardware Architectures . . . 2
- NVIDIA GPU Architectures . . . 2
- AMD GPU Architectures . . . 2

Compiling with CUDA (NVIDIA GPUs) . . . 3
- Prerequisites . . . 3
- CMake Configuration for CUDA . . . 3
- Build Commands . . . 4
- Running on GPUs . . . 4
- Advanced CUDA Options . . . 4

Compiling with HIP (AMD GPUs) . . . 5
- Prerequisites . . . 5
- CMake Configuration for HIP . . . 5
- Build Commands . . . 6
- Running on GPUs . . . 6
- Advanced HIP Options . . . 6

Combining Backends at Compile Time . . . 7
Troubleshooting . . . 7
- CUDA/NVCC Compilation Errors . . . 7
- HIP/ROCm Compilation Errors . . . 7
- CMake Configuration Issues . . . 8

Performance Considerations . . . 8
- Device Selection . . . 8
- Memory Management . . . 8
- Occupancy and Block Sizes . . . 9

Further Resources . . . 9

Overview

Kokkos provides a unified programming model that works across different GPU architectures by abstracting the underlying programming models. NVIDIA GPUs use **CUDA**, while AMD GPUs use **HIP** (Heterogeneous-compute Interface for Portability). This chapter explains how to compile Kokkos code to target these different GPU backends.

Hardware Architectures

Before compiling for a specific GPU, you need to know what hardware you're targeting. The architecture name is crucial for optimal performance.

NVIDIA GPU Architectures

NVIDIA GPUs have evolved through several architecture generations. Some common ones:

Architecture	Kokkos Flag	GPU Examples	Notes
Ampere	`Kokkos_ARCH_AMPERE80`	A100, A10	Latest high-performance GPUs
Ampere	`Kokkos_ARCH_AMPERE86`	RTX 30 series (desktop)	Consumer-grade Ampere
Turing	`Kokkos_ARCH_TURING75`	RTX 20 series, T4	Older data center GPUs
Volta	`Kokkos_ARCH_VOLTA70`	V100, Titan V	Previous generation data center
Pascal	`Kokkos_ARCH_PASCAL61`	GTX 10 series	Compatible with CUDA < 13.0

To check your GPU, use:

```
nvidia-smi
```

AMD GPU Architectures

AMD's data-center GPUs (the Instinct MI series) use the **CDNA** architecture family, while consumer Radeon cards use the **RDNA** family. Kokkos identifies AMD GPUs by their GFX ISA name:

Architecture	Kokkos Flag	GPU Examples
CDNA3	`Kokkos_ARCH_AMD_GFX942`	MI300A, MI300X
CDNA2	`Kokkos_ARCH_AMD_GFX90A`	MI210, MI250, MI250X
CDNA1	`Kokkos_ARCH_AMD_GFX908`	MI100
RDNA3 (consumer)	`Kokkos_ARCH_AMD_GFX1100`	Radeon RX 7900 XTX

To check your AMD GPU and ROCm version, use:

```
rocm-smi
hipcc --version
```

Compiling with CUDA (NVIDIA GPUs)

Prerequisites

Before compiling, ensure you have:

1. **NVIDIA CUDA Toolkit** installed (version 11.8 or later recommended)
2. **NVIDIA GPU drivers** that support your GPU
3. **CMake** build tool (at least version 3.21)
4. A **host C++ compiler** (e.g. `g++`); `nvcc`, the CUDA compiler driver, comes with the CUDA Toolkit

Check your CUDA installation:

```
nvcc --version
```

CMake Configuration for CUDA

The backend (CUDA, HIP, Threads, …) is a **compile-time** decision in Kokkos: it is selected with `Kokkos_ENABLE_*` CMake options when Kokkos itself is built. If you pull Kokkos into your project with CMake's `FetchContent` (as the course skeleton does), you simply pass these options when configuring your own project and Kokkos takes care of invoking the CUDA compiler — you do **not** set `CMAKE_CXX_COMPILER` to `nvcc`.

Alternatively, you can install Kokkos separately. Use version 4.7.03, which is the version this course is pinned to:

```
git clone --branch 4.7.03 https://github.com/kokkos/kokkos.git
cd kokkos
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release \
      -DKokkos_ENABLE_SERIAL=ON \
      -DKokkos_ENABLE_THREADS=ON \
      -DKokkos_ENABLE_CUDA=ON \
      -DKokkos_ARCH_AMPERE80=ON \
      ..
cmake --build . -j $(nproc)
cmake --install . --prefix ~/kokkos-install
```

By default, Kokkos compiles CUDA code through its `nvcc_wrapper` script, which combines `nvcc` with your host C++ compiler. If you want to specify a compiler explicitly, use `-DCMAKE_CXX_COMPILER=<kokkos-source>/bin/nvcc_wrapper` or a `clang++` with CUDA support — but never plain `nvcc`, which cannot act as a general C++ compiler.

Then use Kokkos in your project's `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.21)
project(WaveEquation LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

```
find_package(Kokkos REQUIRED)

add_executable(wave wave.cpp)
target_link_libraries(wave PRIVATE Kokkos::kokkos)
```

Build Commands

Set up and compile with CUDA:

```
# Create build directory
mkdir build
cd build

# Configure with Kokkos location
cmake -DKokkos_DIR=~/kokkos-install/lib/cmake/Kokkos ..

# Compile
cmake --build . -j $(nproc)

# Run on GPU
./wave
```

Running on GPUs

There is no runtime switch to select the backend: because CUDA was enabled at compile time, it is the default execution space and all kernels run on the GPU automatically. A few runtime options exist for tuning:

```
# Run (kernels execute on the GPU)
./wave

# Use a specific GPU (if you have multiple)
./wave --kokkos-device-id=1

# View available options
./wave --kokkos-help
```

Advanced CUDA Options

For different NVIDIA architectures, change the Kokkos architecture flag when configuring Kokkos:

```
# For A100 (data center)
cmake -DKokkos_ARCH_AMPERE80=ON ..

# For RTX 3090 / RTX 4090 (consumer)
cmake -DKokkos_ARCH_AMPERE86=ON ..
```

```
# For V100
cmake -DKokkos_ARCH_VOLTA70=ON ..

# For T4
cmake -DKokkos_ARCH_TURING75=ON ..
```

Compiling with HIP (AMD GPUs)

Prerequisites

Before compiling for AMD GPUs, ensure you have:

1. **AMD ROCm** installed (version 5.0 or later recommended)
2. **AMD GPU drivers** for ROCm
3. **CMake** build tool (at least version 3.21)
4. **hipcc** compiler (comes with ROCm)

Check your ROCm installation:

```
hipcc --version
rocm-smi
```

CMake Configuration for HIP

As with CUDA, the HIP backend is selected at compile time via `Kokkos_ENABLE_HIP`. If Kokkos comes into your project via `FetchContent`, just pass the options below when configuring your project. To install Kokkos (version 4.7.03, as used in this course) separately with HIP support:

```
git clone --branch 4.7.03 https://github.com/kokkos/kokkos.git
cd kokkos
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release \
      -DKokkos_ENABLE_SERIAL=ON \
      -DKokkos_ENABLE_THREADS=ON \
      -DKokkos_ENABLE_HIP=ON \
      -DKokkos_ARCH_AMD_GFX90A=ON \
      -DCMAKE_CXX_COMPILER=hipcc \
      ..
cmake --build . -j $(nproc)
cmake --install . --prefix ~/kokkos-install
```

Then use Kokkos in your project's `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.21)
project(WaveEquation LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

```
find_package(Kokkos REQUIRED)

add_executable(wave wave.cpp)
target_link_libraries(wave PRIVATE Kokkos::kokkos)
```

Build Commands

Set up and compile with HIP:

```
# Create build directory
mkdir build
cd build

# Configure with Kokkos location
cmake -DKokkos_DIR=~/kokkos-install/lib/cmake/Kokkos ..

# Compile
cmake --build . -j $(nproc)

# Run on GPU
./wave
```

Running on GPUs

As with CUDA, the HIP backend is fixed at compile time and becomes the default execution space — kernels run on the GPU without any runtime flag:

```
# Run (kernels execute on the GPU)
./wave

# Use a specific GPU (if you have multiple)
./wave --kokkos-device-id=1

# View available options
./wave --kokkos-help
```

Advanced HIP Options

For different AMD architectures, change the Kokkos architecture flag when configuring Kokkos:

```
# For MI300A / MI300X (latest)
cmake -DKokkos_ARCH_AMD_GFX942=ON ..

# For MI200 series (MI210, MI250, MI250X)
cmake -DKokkos_ARCH_AMD_GFX90A=ON ..

# For MI100
cmake -DKokkos_ARCH_AMD_GFX908=ON ..
```

Combining Backends at Compile Time

Backend selection in Kokkos is a **compile-time** decision — there is no runtime flag to switch backends. A single Kokkos build can enable several *host* backends (Serial, Threads, OpenMP) plus at most **one** *device* backend (CUDA **or** HIP):

```
# Configure Kokkos with several host backends and the CUDA device backend
cmake -DCMAKE_BUILD_TYPE=Release \
      -DKokkos_ENABLE_SERIAL=ON \
      -DKokkos_ENABLE_THREADS=ON \
      -DKokkos_ENABLE_CUDA=ON \
      -DKokkos_ARCH_AMPERE80=ON \
      ..
cmake --build . -j $(nproc)
cmake --install . --prefix ~/kokkos-install
```

The **default execution space** is also fixed at compile time: if a device backend is enabled it takes precedence, otherwise the highest-priority enabled host backend is used. In the build above, `Kokkos::DefaultExecutionSpace` is `Kokkos::Cuda`, and all kernels that do not explicitly request another execution space run on the GPU. The enabled host backends remain available in code, e.g. via `Kokkos::DefaultHostExecutionSpace`. If you want the same program to run *by default* on the CPU instead, you need a separate build with the device backend disabled. Runtime flags only tune behavior within the compiled configuration:

```
# Set the number of threads for the host backend
./wave --kokkos-num-threads=4

# Select a GPU by id
./wave --kokkos-device-id=0
```

Troubleshooting

CUDA/NVCC Compilation Errors

Problem: `nvcc: not found` - **Solution:** Add CUDA to your PATH. If using the default CUDA installation:

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Problem: Architecture mismatch errors - **Solution:** Verify your GPU architecture with `nvidia-smi` and set the correct `Kokkos_ARCH_*` flag when configuring Kokkos with CMake

HIP/ROCm Compilation Errors

Problem: `hipcc: not found` - **Solution:** Load the ROCm module (on HPC clusters):

```
module load rocm
```

Or add to PATH if ROCm is installed locally:

```
export PATH=/opt/rocm/bin:$PATH
```

Problem: "Could not find HIP" - **Solution:** Set the HIP path explicitly in your CMake configuration or environment:

```
export HIP_PATH=/opt/rocm/hip
cmake -DHIP_PATH=/opt/rocm/hip ..
```

CMake Configuration Issues

Problem: "CMake version too old" - **Solution:** Upgrade CMake to at least version 3.21. See notes/CMake.qmd for details.

Problem: CMake not found for Kokkos subproject - **Solution:** Ensure CMake is installed:

```
# Ubuntu/Debian
apt install cmake

# macOS
brew install cmake
```

Performance Considerations

Device Selection

- **NVIDIA A100:** Best for general-purpose HPC, uses `Kokkos_ARCH_AMPERE80`
- **NVIDIA H100:** Latest generation, uses `Kokkos_ARCH_HOPPER90`
- **AMD MI250 (CDNA2):** Best price-to-performance, uses `Kokkos_ARCH_AMD_GFX90A`
- **AMD MI300X (CDNA3):** Latest, highest bandwidth, uses `Kokkos_ARCH_AMD_GFX942`

Memory Management

Both CUDA and HIP support unified memory, but explicit device transfers via `Kokkos::deep_copy` often provide better performance:

```
// Allocate on device
Kokkos::View<double*> device_view("device", N);

// Copy from host to device
Kokkos::deep_copy(device_view, host_view);

// Copy from device to host
Kokkos::deep_copy(host_view, device_view);
```

Occupancy and Block Sizes

For optimal GPU utilization, consider: - GPU memory bandwidth vs. computation ratio - Thread block size (typically multiples of 32 for NVIDIA, 64 for AMD) - Register pressure and shared memory usage

Kokkos handles many of these automatically, but understanding these concepts helps optimize your kernels.

Further Resources

- Kokkos Documentation - Compiling
- Kokkos Configuration Guide
- NVIDIA CUDA Documentation
- AMD ROCm Documentation
- HIP Programming Guide