Accelerator-Based Programming
-
GPU programming with Kokkos

Jörn Zimmerling

September 26, 2022

UPPSALA
UNIVERSITET

Programming of accelerators

- Many big research codes target a variety of platforms
  - Classical CPU-based systems (Intel, AMD, arm)
  - NVIDIA GPUs
  - AMD GPUs
  - Intel GPUs
  - …
- How to address difference in architectures?
  - Separately implementing everything on multiple architectures not realistic
    - Write and maintain code with multiple paths - `#ifdef`
    - Needs highly skilled people on several architectures
  - Separately implementing important algorithms on multiple architectures works sometimes
    - Needs big enough team to drive it
  - Separation of concerns important: Multiple “Back-ends” (CPU,GPU) for key algorithms, but abstract it away from application code
- Create abstraction layer that targets generic loops

Kokkos

- Targets a single-source implementation with C++ programming language
- Create execution policy to important computational kernels (for loop, reduction, . . .) and express work as computational body (code to perform unit of work)
- Descriptive programming model
- Compile the code for different targets: CPU/GPU
- Select different data layouts adapted to hardware architecture
  - Map algorithm parallelism to hardware parallelism
  - Granularity of data access from different threads
- Goals of Kokkos:
  - Ease of use
  - Flexibility also to large codes
  - Performance

→ Performance portability

Kokkos in a nutshell

- Model for data parallelism
  - Use **parallel patterns** and **execution policies** to execute **compute bodies**
  - Example 1: parallel loops with the `parallel_for` pattern (e.g. vector add)

```
parallel_for ( ”stream_triad” , N , [=] (int i) {
        z( i) = a * x( i) + y( i) ;
        }) ;
```

  - `[=]` – lambda capture list (`[=]` value, `[&]` reference) `[c++ 11]`
  - Example 2: Reductions combine results from loop iterations

```
float result;
parallel_reduce ( ” summation ” , N ,[=] ( int i, float &lres ) {
                lres += x( i) ;
                }, result) ;
```

  - Flexibility choose data-layout problem with multi-dimensional array abstraction
  - Execution and memory spaces to control
    - Where data lives
    - Where code executes

Learning Kokkos

- We will make use of extensive tutorial material provided by
- Kokkos team
  - General Kokkos GitHub project organization
  - https://github.com/kokkos
  - Lecture material: https://github.com/kokkos/kokkos-tutorials/tree/main/LectureSeries
- We will discuss
  - Lecture 1: Introduction
  - Lecture 2: Memory views and spaces
  - Lecture 3: Multi-dimensional loops and data structures
- Get Kokkos from https://github.com/kokkos/kokkos
- Lab consist out of tutorial’s from Kokkos