Karlsruhe Institute of Technology

Scientific Computing Center

A comparison of GPU offloading techniques

Holger Obermaier | 24 July 2024

1. Overview

- Example Program - GPU Offloading workflow

2. GPU offloading based on compiler pragmas

- OpenMP (Open Multi-Processing)
- OpenACC (Open Accelerators)

3. GPU offloading based on programming language extensions

- C++ Standard Parallelism
- SYCL
- CUDA (Compute Unified Device Architecture)
- HIP (Heterogeneous-Compute Interface for Portability)

4. GPU offloading based on libraries

- OpenCL (Open Computing Language)
- Kokkos

Overview

- Why offloading to GPUs?
  - Dedicated fast memory (e.g. HBM)
  - Many parallel execution units
  - The majority of HoreKA's computing power comes from GPUs
- Many techniques for GPU offloading
  - Compiler pragmas
  - Programming language extensions
  - Libraries
- No clear winner
- Comparison based on
  - Usability, simplicity
  - Achievable performance
  - Supported compilers
  - Hardware portability

Example Program

GPU Offloading workflow

- Retrieve platform information
- Allocate host memory
- Pre-process / initialize data on the host (e.g. read data from storage)
- Allocate device memory
- Transfer data: Host memory $\rightarrow$ Device memory
- Compute on the device
- Transfer data: Device memory $\rightarrow$ Host memory
- Free device memory
- Post-process data on the host (e.g. write data to storage)
- Free host memory

OpenMP (Open Multi-Processing)

Overview

- Directive-based parallel programming model for C, C++ and Fortran
- Originally only targeted shared-memory multiprocessing
- GPU offload support added more recently
- Managed by nonprofit corporation *OpenMP Architecture Review Board*

Supported Compilers

- GCC
- Intel oneAPI Compiler
- LLVM
- NVIDIA HPC SDK Compiler

Hardware portability

- CPUs
- AMD GPUs
- Intel GPUs
- NVIDIA GPUs

OpenACC (Open Accelerators)

Overview

- Directive-based parallel programming model for C, C++ and Fortran
- Launched before OpenMP provided GPU offloading support ⇒ Focus on accelerators
- Many of the OpenACC concepts have since been incorporated into OpenMP
- Managed by the nonprofit OpenACC Organization

Supported Compilers

- GCC (OpenACC 2.6 from 2017)
- NVIDIA HPC SDK Compiler

Hardware portability

- CPUs
- NVIDIA GPUs

C++ Standard Parallelism

Overview

- C++17 introduced parallel algorithms, extended in C++20
  - Includes parallel loops operations e.g. `for_each` and `transform_reduce`
  - Execution policies (`seq`, `par`) give compiler hints
  - Single source code for CPU and accelerator
- No explicit data placement / device selection
- Execution can be serial! Parallel execution on CPUs or GPUs needs compiler support

Supported Compilers

- GCC (CPU only)
- Intel oneAPI Compiler (CPU only)
- LLVM (CPU only)
- NVIDIA HPC SDK Compiler

Hardware portability

- CPUs
- NVIDIA GPUs

SYCL

Overview

- Higher-level programming model (APIs, ecosystem)
- Provides APIs to find devices, to manage data resources and code execution on those
- Standard C++, single source code for CPU and accelerator
- SCYLomatic: CUDA to SYCL converter
- developed by Khronos Group

Supported Compilers

- Intel oneAPI Compiler

Hardware portability

- CPUs
- AMD GPUs (Codeplay Plugin)
- Intel GPUs
- NVIDIA GPUs (Codeplay Plugin)

CUDA (Compute Unified Device Architecture)

Karlsruhe Institute of Technology

Overview

- Collection of accelerated libraries and extensions for C, C++ and Fortran
- Low-level programming model, full control on data placement and code execution
- Kernels (device code) can not run on host CPUs
- CUDA code is not C/C++/FORTRAN compliant ⇒ Compiling requires NVIDIA or LLVM compiler
- Proprietary software, closed source
- Available for a long time ⇒ Most probably market leader
- Comprehensive solution (e.g cuBLAS, cuFFT)

Supported Compilers

- LLVM
- NVIDIA HPC SDK Compiler

Hardware portability

- NVIDIA GPUs

HIP (Heterogeneous-Compute Interf. for Portability)

Karlsruhe Institute of Technology

Overview

- 1 to 1 CUDA clone, e.g. `cudaMalloc` ⇒ `hipMalloc`
- Not all CUDA features and libraries are available
- `hipify-clang` / `hipify-perl`: LLVM / Regex based CUDA to HIP converter
- Open source (MIT License)

Supported Compilers

- AMD ROCm Compiler
- LLVM

Hardware portability

- AMD GPUs (ROCm backend)
- NVIDIA GPUs (CUDA backend)

OpenCL (Open Computing Language)

Overview

- OpenCL is a *low-level* programming framework
- Full control on data placement and code execution
- Support for multiple heterogeneous types of execution resources
- Host code is written in C or C++, GPU code is written in OpenCL C ($\sim$ C99)
- Open standard maintained by non-profit technology consortium Khronos Group

Supported Compilers

- All C, C++ compiler

Hardware portability

- CPUs
- AMD GPUs
- Intel GPUs
- NVIDIA GPUs

Kokkos

Overview

- Programming model in C++ for performance portable applications
- Abstractions for both parallel code execution and data management
- Open Source, Linux Foundation project

Supported Compilers

- All C++ compiler

Hardware portability

- CPUs (OpenMP backend)
- AMD GPUs (HIP backend)
- Intel GPUs (SYCL backend)
- NVIDIA GPUs (CUDA backend)