A Performance Analysis of Modern Parallel Programming Models Using a Compute-Bound Application

Duy Dung Nguyen

25.01.2022

Seminar Supercomputer: Forschung und Innovation

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg
Table of contents

1. Introduction
   • Modern parallel programming models
   • miniBUDE as a benchmark

2. Performance Analysis

3. Results of the performance analysis
   • CPUs
   • GPUs

4. Towards achieving performance portability / Summary
Introduction
Modern Parallel Programming Models

- A set of program abstractions for fitting parallel activities from the application to the underlying parallel hardware.
- Shared memory and threads.
- The C++ language.
- OpenMP (for CPU and offload), OpenCL, CUDA, OpenACC, Kokkos and SYCL.
Performance portability

• What is Performance portability?
→ “Performance portability means the same source code will run productively on a variety of different architectures" (Larkin)
• Portability is a common concern for developers—and users—of modern programming models.
miniBUDE as a benchmark

- A mini-app created from the core computation kernel for BUDE (Bristol University Docking Engine).
- Each subdirectory contains a separate C/C++ implementation.
miniBUDE is a compute-bound application

### Compute-bound (CPU bound)
- means the rate at which process progresses is limited by the speed of the CPU.
- Example: multiplying small matrices.

### Memory-bound
- means the rate at which a process progresses is limited by the amount memory available.
- Example: multiplying large matrices.
Performance Analysis
Hardware platforms and compilers used

- Aggressive compiler optimization flags to the level of 
  \(-march=native -Ofast\).
<table>
<thead>
<tr>
<th>Compiler</th>
<th>CPUs</th>
<th>GPUs</th>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOCC 2.3</td>
<td>X</td>
<td></td>
<td>m k s</td>
</tr>
<tr>
<td>AOMP 11.0</td>
<td></td>
<td>M</td>
<td>m</td>
</tr>
<tr>
<td>Arm Compiler 21.0</td>
<td>R</td>
<td></td>
<td>m k s</td>
</tr>
<tr>
<td>ComputeCpp 2.1.1</td>
<td>X</td>
<td>I</td>
<td>m k s</td>
</tr>
<tr>
<td>Cray Compiler 10.0</td>
<td>R X</td>
<td>N</td>
<td>a&lt;sup&gt;1&lt;/sup&gt; m k s</td>
</tr>
<tr>
<td>Fujitsu Compiler 4.3</td>
<td>R</td>
<td>X</td>
<td>m k s</td>
</tr>
<tr>
<td>GCC 10.3</td>
<td>R X</td>
<td>M N</td>
<td>a&lt;sup&gt;1&lt;/sup&gt; m k s</td>
</tr>
<tr>
<td>Intel ICX 2019</td>
<td>X</td>
<td></td>
<td>m k&lt;sup&gt;2&lt;/sup&gt; s</td>
</tr>
<tr>
<td>Intel DPC++ 2021.1</td>
<td>X</td>
<td>N</td>
<td>m k s</td>
</tr>
<tr>
<td>LLVM 11.0</td>
<td>R X</td>
<td>N</td>
<td>m k s</td>
</tr>
<tr>
<td>NVCC 10.2</td>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>PGI 19.10</td>
<td></td>
<td>N</td>
<td>a</td>
</tr>
</tbody>
</table>

CPUs: ARM, X86; CPUs: AMD, NVIDIA, INTEL
Frameworks: cuda, openacc, opencl, openmp, kokkos, sycl

<sup>1</sup> Version 9.0 only; <sup>2</sup> With the experimental INTEL.GEN backend.
Performance Analysis

• On CPU platforms, hardware counters were accessed through the built-in Linux `perf` tool.
• On GPUs, NVIDIA CUDA profiler and the OpenCL intercept layer were used.
Results of the performance analysis
OpenMP, Kokkos und SYCL

CPUs
OpenMP - Introduction

• An Application Program Interface (API)
• Gives parallel programmers a simple and flexible interface for developing portable parallel applications.
• The API is specified for C/C++ and Fortran
• API components:
  ➢ Compiler Directives
  ➢ Runtime Library Routines
  ➢ Environment variables
OpenMP - Performance on CPUs

- There is no OpenMP implementation that works optimal on every single platform.
- AMD seems to be a good choice.
OpenMP - Performance on CPUs

- Cache-aware roofline for the Cascade Lake platform showing the achieved performance for miniBUDE.
Kokkos - Introduction

- Parallelism expressed via the idiomatic Kokkos:: parallel_for function.
  - ParallelFunctor functor;
    Kokkos::parallel_for(numberOfIterations, functor);
- A C++ compiler is the only requirement.
- Kokkos lets you write algorithms once and run on many architectures, including both CPUs and GPUs.
Kkokkos – Performance on CPUs

- The results shows a strong correlation compared to the OpenMP implementation results.
SYCL - Introduction

• Enables code to be written in a “single-source” style using completely standard C++.
• The kernel is a direct port of the OpenCL version.
• For comparison, a separate kernel that is closer to the OpenMP was implemented.
SYCL - Performance on CPUs

- Only results on platforms where at least two implementations were supported.
- The Skylake platform is missing from these results.
Open CL, CUDA, OpenMP Offload, Open ACC, Kokkos und SYCL

GPUs
Low-level: OpenCL and CUDA

OpenCL
• A framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs and other processors.

CUDA
• Is a parallel computing platform and programming model developed by Nvidia for general computing on its own GPUs.
Low-level: OpenCL and CUDA

• On the NVIDIA V100: the CUDA implementation was 18% faster than the OpenCL code.
• Both versions also ran on the AMD Radeon VII, but OpenCL was 1.6× faster on this platform.
• CUDA and HIP cannot be used on the Intel GPU.
Directives-Based: OpenMP Offload and OpenACC

OpenMP Offload
• In OpenMP API 4.0, the specification provides a set of directives to instruct the compiler to offload a block of code to the device.

OpenACC
• A programming standard for parallel computing on accelerators (mostly on NVIDIA GPU).
Directives-Based: OpenMP Offload and OpenACC

- Virtually identical performance on the V100.
- The directives-based approach showed about $0.4 \times$ the performance of the optimized CUDA code.
- On the Radeon, OpenACC was two orders of magnitude slower than OpenMP, which in turn only reached $0.3 \times$ the performance of the fastest model, OpenCL.
- On the Intel GPU, OpenMP target reached only $0.2–0.3 \times$ the performance of the fastest model (SYCL).
High-level: Kokkos and SYCL

- Kokkos and SYCL both run on all the GPUs studied, but only one implementation, hipSYCL, runs on AMD and NVIDIA.
- Kokkos: the code run on the GPU platforms was unchanged from the version run on CPUs.
Performance of the GPU implementations

- A direct performance comparison between these platforms is not useful.
- CUDA (V100) = 2× OpenCL (RadeonVII) = 14× SYCL (Iris Pro580)
Towards achieving performance portability - 
Summary
Achieved performance across all programming models

- No programming model can currently achieve optimal performance on all platforms.
- Kokkos was the only framework that was able to support all CPU and GPU platforms in one package.
Towards achieving performance portability

- Performance was lower with hipSYCL compared to Kokkos or plain OpenMP.
- Portability between CPUs and GPUs remains a concern.
- When running several iterations of a benchmark, the first run was usually up to 2× slower than subsequent runs.
Summary

- True performance portability is still out of reach.
- On GPUs, low-level APIs continue to provide the highest possible performance.
- Kokkos emerged as a reliable choice, and OpenMP remains in a strong position.
References


