Developing High-Performance, Portable OpenCL Code via Multi-Dimensional Homomorphisms

Ari Rasch
a.rasch@wwu.de
University of Muenster
Muenster, Germany

Richard Schulze
r.schulze@wwu.de
University of Muenster
Muenster, Germany

Sergei Gorlatch
gorlatch@wwu.de
University of Muenster
Muenster, Germany

CCS CONCEPTS

• **Computing methodologies → Parallel computing methodologies**; • **Computer systems organization → Parallel architectures**.

KEYWORDS

OpenCL, Performance-Portability, Multi-Dimensional Homomorphisms, Auto-Tuning, GPU, multi-core CPU, BLAS, Stencil

ACM Reference Format:
Ari Rasch, Richard Schulze, and Sergei Gorlatch. 2019. Developing High-Performance, Portable OpenCL Code via Multi-Dimensional Homomorphisms. In *International Workshop on OpenCL (IWOCL'19), May 13–15, 2019, Boston, MA, USA.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3318170.3318171

1 ABSTRACT

A key challenge in programming high-performance applications is achieving portable performance, such that the same program code can reach a consistent level of performance over the variety of modern parallel processors, including multi-core CPU and many-core Graphics Processing Units (GPU), and over the variety of problem sizes.

Popular approaches to parallel programming are either restricted to the hardware of a particular vendor (like CUDA for NVIDIA) or, even if they provide code portability (like OpenCL), performance portability is usually not available: for example, a parallel program achieving high performance on a GPU often yields poor performance on a CPU, or even on another GPU model. The reason is that hardware architectures differ significantly in their characteristics, e.g., GPU provide a high number of cores but small caches while CPU have a low number of cores and big caches; also GPU from different vendors (e.g., NVIDIA vs. AMD) pose different or even contradicting requirements on the code for achieving the full performance potential of the corresponding architecture. Performance differs also across input sizes. For example, a high-performance implementation of GEneral Matrix-Matrix Multiplication (GEMM) targeting big input matrices differs significantly from a GEMM implementation optimized for small matrices, e.g., as used in deep learning. This is because high performance on big matrices is achieved by computing all elements of the resulting matrix simultaneously and each of them sequentially, whereas for high performance on small matrices, the computation of each element should be parallelized as well.

The lack of performance portability often requires re-designing program code for every new target architecture and/or another problem size.

In this talk, we address an approach to performance portability based on patterns of parallelism and auto-tuning. We extend the functional formalism of Multi-Dimensional Homomorphisms (MDH) that allows expressing a wide range of applications (including the popular BLAS routines and stencil computations) as MDH-instances. For MDH, we develop a generic OpenCL implementation schema. This schema is performance-portable: it is parametrized with the performance-critical parameters of OpenCL's platform and memory model, such that, for each particular MDH-instance, particular problem size and particular target architecture, we can fully automatically find the well-performing parameter values using our novel Auto-Tuning Framework (ATF), and thereby adapt the OpenCL code correspondingly.

Our experiments with linear algebra routines (BLAS) and stencil applications demonstrate that we reach competitive and often even significantly better performance than the related work – e.g., speedup factors of up to 5x over the hand-implemented, vendor-provided BLAS libraries Intel MKL and NVIDIA cuBLAS – on representative parallel architectures and for important input sizes that are used in deep learning.

IWOCL'19, May 13–15, 2019, Boston, MA, USA

ACM ISBN 978-1-4503-6230-6/19/05.
https://doi.org/10.1145/3318170.3318171