Enhancing spatial parallelism on loop structure for FPGA

Yuka Sano, Taisuke Boku,
Norihisa Fujita, Ryohei Kobayashi
CCS, University of Tsukuba
Japan

Mitsuhisa Sato, Miwako Tsuji
R-CCS, RIKEN
Japan

EXTENDED ABSTRACT

In today's HPC systems, GPUs with high computational performance and memory bandwidth are the leading players. However, GPU-based acceleration is designed to excel when utilizing many computation cores and performing SIMD/STMD manner of synchronized computation over a large number of uniform data array elements. Therefore, it may not fully exploit its computational performance in calculations with low parallelism, complex operations involving conditional branching, or parallel applications with frequent inter-node communication to interrupt continuous computing on GPU devices. One of the alternative solutions for accelerated computing is FPGA (Field Programmable Gate Array), especially with recent advancements in devices containing a large number of logic elements, high memory bandwidth, and even multiple channels of high-speed optical interconnection interfaces, reaching up to 100 Gbps for each. The performance of an FPGA is based on pipeline parallelism, enabling the computation stream to continue even with conditional branches.

Currently, it is available to program FPGA devices in high-level languages such as OpenCL. However, the programmer needs high optimization skills to exploit its potential performance. To solve this problem of FPGA utilization for HPC applications, we have been developing an OpenACC-ready compiler for FPGA. There are several kinds of research on this target, such as OpenARC[2] research compiler by ORNL. However, we focus more deeply on the automatic performance optimization on the compiler level. This research has been performed based on Omni OpenACC compiler[3] in collaboration with the Center for Computational Sciences at the University of Tsukuba (CCS) and RIKEN Center for Computational Science (R-CCS).

In this study, we evaluate and examine high-level synthesis-based FPGA programming techniques towards the compiler-based performance optimization. At first, we examine several optimization techniques expressed in OpenCL for Intel FPGA. Then, we modify the current Omni OpenACC compiler for OpenACC-to-OpenCL generation for optimized OpenCL code output for FPGA execution. Specifically, we try various techniques to increase the number of computational elements by spatial parallelism, such as pipelining, loop unrolling, and simultaneous execution of multiple kernels. Here we target the CG (Conjugate Gradient) method code for matrix calculation described in OpenCL.

We vary the loop unroll factor and the number of kernels executed simultaneously to evaluate the FLOPS and BRAM (Block RAM) usage. BRAM is a kind of private SRAM owned by each computation kernel and not shared over multiple kernels. We investigate the impact of increasing the number of operations within a clock cycle on BRAM usage and explore strategies for performance optimization. However, in FPGA, the depth of loop unrolling, the number of used functional units, and memory usage affect the operating frequency, leading to complex trade-offs. When performing loop unrolling, attention must be paid to data propagation dependencies in order not to increase the initiation interval (II), which indicates how many clock cycles each stage of the pipeline completes. Moreover, when the loop unrolling depth increases, the OpenCL compiler for FPGA automatically generates a copy of BRAM contents. Therefore, the BRAM capacity limits the performance gain by loop unrolling depth.

Another challenge is to distribute the workload to simultaneous multiple kernels with the domain decomposition method. However, this approach introduces kernel-to-kernel communication[1], and when the number of kernels increases, the operating frequency may decrease significantly. Therefore, it is necessary to appropriately combine loop unrolling and simultaneous execution of multiple kernels depending on the execution environment. Here we apply a technique to create multiple kernels such as distributed memory processes in PGAS model.

Based on the optimization methods obtained in this research, we are implementing the functionality to generate OpenCL code optimized for FPGAs from OpenACC using the Omni OpenACC compiler. We have already implemented loop unrolling and profilers for data transfer time and kernel's execution time in the compiler. This feature will provide existing FPGA programmers with a more straightforward programming environment than OpenCL. Additionally, the programming approach of adding directives to sequential code is expected to reduce the amount of code and development time. Furthermore, FPGA acceleration efforts are expected to expand to applications that have been reluctant to use FPGA-based acceleration until now.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI (Grant Number 21H04869) and the MCRP 2023 Program by the Center for Computational Sciences, University of Tsukuba. We also thank the Intel University Program for providing the hardware and software.

REFERENCES

[1] Intel FPGA SDK for OpenCL. 2022. 3.5. channels. Intel. Available: https://www.intel.com/content/www/us/en/docs/programmable/683521/22-4/channels-opencl-architectural-viewer.html. [Online].

[2] Seyong Lee and Jeffrey S. Vetter. 2014. OpenARC: Open Accelerator Research Compiler for Directive-Based, Efficient Heterogeneous Computing. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing* (Vancouver, BC, Canada) *(HPDC '14)*. Association for Computing Machinery, New York, NY, USA, 115–120. https://doi.org/10.1145/2600212.2600704

[3] Akihiro Tabuchi, Yasuyuki Kimura, Sunao Torii, Hideo Matsufuru, Tadashi Ishikawa, Taisuke Boku, and Mitsuhisa Sato. 2016. Design and preliminary evaluation of Omni OpenACC compiler for massive MIMD processor PEZY-SC. In *OpenMP: Memory, Devices, and Tasks*, Vol. 9903. 293–305. https://doi.org/10.1007/978-3-319-45550-1_21