Bachelor Thesis

Automatic CPU-FPGA Hybrid Execution of Scientific Benchmarks

Student: Noah Berner

Tobias Grosser, David Cock, Timothy Roscoe
ETH Zurich

Introduction

Today computational performance increases faster than memory performance. Consequently, memory bandwidth will in the future be the primary bottleneck. The use of directly wired connections as a replacement for expensive memory operations, as available in data-flow platforms such as FPGAs, has the potential to reduce the cost of such data transfers. By using specialized tensor cores NVIDIA's Volta GPU successfully reduced register transfers and Intel provides first hardware that closely integrates FPGAs into their general-purpose CPUs.

A combination of data-flow and general-purpose compute devices is needed to execute full programs. Data-flow architectures are effective in case the computation can be placed statically and independent of the input data, but for code that is heavy in dynamic control traditional microprocessors are still needed. Hence, we envision for the future an increased use of hybrid architectures where suitable computational kernels are offloaded to data-flow based accelerators much like SIMD units are used in today's architectures.

The effective compilation of full applications to such devices is necessary for them to have broad impact. Today high-level synthesis tools allow for the automatic synthesis of computational kernels to FPGA code, but the automatic compilation of full scientific benchmarks to future hybrid data-flow architectures remains hard. Enzian [1] provides the perfect hardware platform to study the behaviour of such a hybrid architecture, as it closely combines CPU and FPGA to a single shared-memory system. However, Enzian lacks the software environment to automatically port code to this new architecture.

Existing solutions

For systems which use GPU accelerators instead of FPGAs, a solution is to use Polly-ACC [2], a compiler which automates the mapping onto GPU hybrid architectures. A great number of HLS tools exist, for example Vivado High-Level Synthesis [3], that allow a user to compile high-level source code to hardware circuits on FPGAs [4]. HLS tools are great for fast design exploration and accessible hardware generation, but most of them do not create code or hardware circuits for a CPU-FPGA hybrid architecture.

There are a few compliers that do generate code for hybrid architectures, like CHiMPS [5] and LegUp [6]. CHiMPS compiles ANSI-C to the Xilinx Accelerated Computing Platform (ACP), a hybrid CPU-FPGA platform, that was still under development when the paper was published in 2008. While there were performance gains when using CHiMPS, the practical use of it was limited, as the ACP never caught on. LegUp uses a different strategy by configuring part of the FPGA to be a RISC CPU and thus creating a CPU-FPGA hybrid architecture on one FPGA.

Objectives and goals

The main goal of the thesis is to get closer to a solution for easy programming of a CPU and FPGA hybrid architecture. Firstly, the two SPEC-benchmarks lbm [7] and cactusADM [8] are analysed when running on a traditional CPU. This serves as a performance baseline and to find the data movements and data size for later FPGA design decisions. Polly-ACC can also be used to get a baseline for a system running a GPU accelerator. In a second phase, the compute kernels of the benchmarks are converted into a hardware design on an FPGA using HLS. Several designs are explored, especially which data is stored where (Registers, BRAM/UltraRAM). Depending on whether or not the Enzian data architecture allows it, a next step is to run the two benchmarks completely on Enzian and to optimize the data transfer between the CPU and the FPGA. As a stretch goal, it is possible to start automating the process for similar programs so that they can be compiled to run on Enzian.

Work Plan

1. An static and CPU-runtime analysis of the two SPEC benchmarks lbm and cactusADM with a focus on memory behavior (e.g., the number of arrays involved, the data volume moved, the volume of read-only data, and other metrics) and other FPGA-relevant characteristics.
2. Architect a design of the two applications that enables efficient FPGA execution.
3. Develop a high-level synthesis implementation of the two benchmarks to run on FPGA (using OpenCL to for coarse-grained data transfer between CPU and FPGAs).
4. Optimize the benchmarks to maximize FPGA performance (e.g., exploit BRAM, UltraRAM, ...).
5. Run applications on an off-the-shelf FPGA and provide a detailed performance, including the overhead of data-transfers.
6. Port applications to Enzian (assuming a suitable HLS software is available for Enzian at this point).
7. Exploit Enzian's fine-grained data-transfer support to accelerate the data-transfers to the FPGA (to a degree the Enzian memory system supports at this point).
8. (optional) Automate this translation in the context of LLVM and Polly.

Deliverables

The project should result in the following concrete deliverables:

1. Thesis dissertation. This should follow the normal format for such a thesis, and contain at least the following:
   - introduction / motivation of the problem
   - discussion of background
   - thorough analysis of related work
   - discussion of the performed performance analysis
   - description of the resulting implementation including its design
   - experimental evaluation
2. Complete source code for:
   - resulting prototype
   - benchmarking/data analysis scripts and corresponding results

   The resulting code should also be accompanied with enough documentation to allow complete reproduction of the experimental results in the paper.
3. Presentation of thesis results and demonstration of functionality.

Grading Scheme

The minimum requirements for a grade of 5.0 are as follows:

- Successful completion of work items 1–4.
- Production of deliverables 1–3 to a satisfactory standard.

The grade will be reduced if these goals are not achieved, except in the case of extreme extenuating circumstances (such as an unforeseeable and unresolvable technical barrier to completing the work, accompanied by an acceptable alternative work item).

A grade of 5.50 or higher will be awarded for the completion of the work to an unusually high quality, the addition of extra work (e.g., large degree of automation), or a particularly novel approach to the problem (such as might result in a workshop publication).

References

[1] Cock et al. (24 February 2019). *Why Enzian?* Retrieved from `http://www.enzian.systems/why-enzian.html`

[2] Grosser, T., & Hoefler, T. (2016). Polly-ACC Transparent compilation to heterogeneous hardware. *2016 International Conference on Supercomputing*, ACM, New York, NY, USA, Article 1, 13 pages. doi:10.1145/2925426.2926286

[3] Xilinx Inc. (26 February 2019). *Vivado High-Level Synthesis: Accelerates IP Creation by Enabling C, C++ and System C Specifications.* Retrieved from `https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`

[4] Meeus, W., Van Beeck, K., Goedemé, T., Meel, J. & Stroobandt, D. (2012). An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems, 16*(3), 31-51. doi:10.1007/s10617-012-9096-8.

[5] Putnam, A., Bennett, D., Dellinger, E., Mason, J., Sundararajan, P. & Eggers, S. (2008). CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures. *2008 International Conference on Field Programmable Logic and Applications*, Heidelberg, 2008, (pp. 173-178). doi:10.1109/FPL.2008.4629927.

[6] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Czajkowski, T., Brown, S. & Anderson, J. (2013). LegUp: An Open-Source High-Level Synthesis Tool for FPGA-Based Processor/Accelerator Systems. *ACM Transactions on Embedded Computing Systems (TECS), 13*(2). doi:10.1145/2514740.

[7] Thomas, P. (23 January 2008). *470.lbm: SPEC CPU2006 Benchmark Description File.* Retrieved from `https://www.spec.org/cpu2006/Docs/470.lbm.html`

[8] Malcolm, T. (16 August 2011). *436.cactusADM: SPEC CPU2006 Benchmark Description.* Retrieved from `https://www.spec.org/cpu2006/Docs/436.cactusADM.html`

Timeline and To-Do List

- **February 2019:**
  Hand in proposal.
- **March 2019 (50%):**
  Start analysis of benchmarks on CPU,
  Do one benchmark first, then the other (around 1 month each for analysis),
  Start writing background section, for example infos about benchmarks, how HLS works and so on.
- **April 2019 (50%):**
  Work on analysis on CPU, find data movement and size
  Start thinking about design on FPGA,
  Write down FPGA design specific info into background section.
- **May 2019 (50%):**
  Finish analysis of benchmarks on CPU,
  Write down the findings of the CPU analysis,
  Use HLS to compile kernel into bitstream for FPGA,
  Analyse kernel performance on FPGA,
  move data into different places (Register, BRAM/UltraRAM).
- **June 2019 (100%):**
  Start of Lernphase (01 June 2019)
  Finish analysis of benchmarks on FPGA,
  Write down the findings of the FPGA analysis,
  Try running the whole benchmark on CPU and FPGA,
  Port the running program to Enzian and figure out data-transfer between CPU and FPGA.
- **July 2019 (100%):**
  Reserve two weeks for studying for exams,
  In the other two weeks, finish Enzian port,
  Write down Enzian related findings.
- **August 2019 (100%):**
  Prüfungsession (05 - 30 August 2019) with two core subject exams,
  Reserve two weeks for studying for exams and writing them,
  Get feedback on current thesis and improve it,
  Hand in thesis at 12 August 2019 ($\sim$3 weeks before actual deadline).
- **September 2019 (100%):**
  Hold presentation before 11 September 2019 (Notenkonferenz).
- **Write thesis in parallel to analysis and other work.**