A Trace-based Workflow for Evaluating Application-specific Memory Bandwidth for FPGA-SoCs

Matthias Göbel*,1, Ahmed Elhossini*,1,
Ben Juurlink*,1

* *Embedded Systems Architecture (AES), Technische Universität Berlin, Germany*

ABSTRACT

FPGA-SoCs such as Xilinx's Zynq-7000 and Altera's Cyclone V SoC provide a great platform for HW/SW-Codesigns. These devices combine a powerful embedded processor with programmable logic similar to that found in FPGAs. For the communication between both parts, FPGA-SoCs provide various interfaces in the logic component that offer access to the DDR memory used by the processor. While high throughput inside an FPGA can be easily achieved for many applications, these interfaces and therefore the memory bandwidth limit the overall performance. This is especially troublesome as it is often challenging to define the requirements in advance. Furthermore, the actual achievable memory bandwidth depends on many parameters like the access pattern.

In this work, we present a workflow that allows to estimate the required memory bandwidth by analyzing the memory trace of an equivalent pure software solution written in C. The workflow also includes mechanisms to simulate the behavior of a HW implementation by mimicking its memory accesses and measure the achieved bandwidth. Thus, it can be determined whether the effort of implementing a HW/SW-Codesign can be justified.

KEYWORDS: FPGA-SoC, Zynq, Cyclone SoC, FPGA, Memory Bandwidth, HW/SW-Codesign

1 Introduction and Related Work

The decision to convert a pure software implementation of an application into a HW/SW-Codesign is often made in the hope of achieving a higher performance and/or a lower energy consumption. However, without a previous analysis of the bandwidth between hardware and software parts and especially the bandwidth to memory, the possible speedup cannot be estimated. In many cases this analysis is not performed at all, resulting in a low speedup or even a negative speedup, i.e. the HW/SW-Codesign is slower than the pure software solution due to certain bottlenecks.

In this work, we present a workflow that estimates the required memory bandwidth and checks whether a specific FPGA-SoC can provide this bandwidth. The workflow starts with measuring the required memory bandwidth of those parts of the application that should be

[1]E-mail: {m.goebel, ahmed.elhossini, b.juurlink}@tu-berlin.de

moved to hardware. This is done by analyzing a pure software solution running on a system that provides the required performance (e.g. the performance required for real-time decoding of a specific video stream). The system could be a powerful desktop PC or a server. In this context, the workflow also creates a trace of all the memory accesses performed by the application. Using this trace, the memory access behavior of an equivalent hardware solution is simulated by performing a HW-optimized version of the same memory accesses in the FPGA. By comparing the measured bandwidth of the pure SW solution and the simulated, and therefore estimated, bandwidth of a hypothetical hardware implementation, the user can decide on whether to implement a HW/SW-Codesign.

Commercial tools such as Xilinx's *SDSoC* allow the estimation of the obtained speedup when implementing functions in hardware. However, it is not clear whether this actually takes the memory accesses into account. Several publications provide information about the available memory bandwidth on FPGA-SoCs [GCAMJ15][SWWB13][SSSS15]. While this information can be very useful for estimating the potential speedup of a HW/SW-Codesign, it does not offer insight into the required memory bandwidth. Furthermore, in-depth knowledge of the memory access pattern of the concerned parts of the application are required as this has a huge impact on the memory bandwidth. These issues will be addressed in this work.

2 The Workflow

Our workflow is based on two assumptions: (i) that BRAM buffers are located between the DDR memory and the hardware accelerator for storing input and output data and (ii) that the actual processing of the data in the accelerator takes no time. The first assumption is justified as it is a common approach for high-throughput HW implementations. The buffers allow to use bursts when reading and writing data from or to DDR memory and thus increase the bandwidth significantly. This approach is also used by Xilinx's high-level synthesis tools (*Vivado HLS*) for converting C to RTL. The second assumption is justified by the high degree of parallelism that an FPGA offers. As a result, the latency, in number of cycles, can often be reduced to almost arbitrarily small values. Furthermore, it is impossible to estimate the actual latency of an implementation without further knowledge about the implementation itself. Finally, as the processing time is significantly shorter than the time the memory accesses take, an assumed latency of zero is a reasonable assumption.

An overview of the workflow can be seen in Figure 1. After compiling the original C application, a memory trace of the complete application is generated using the *Valgrind* profiling and analysis suite. Afterwards, the executable is disassembled in order to get the instruction addresses of the C function that should be implemented in hardware. By using this information, only the memory accesses in this function are extracted out of the complete memory trace. In subsequent steps, all memory accesses that target arrays that are not shared between this function and the remaining application are being removed. This is justified by the fact that local arrays in a function would be moved to BRAM buffers in hardware implementations. Only shared arrays must be located in DDR memory. Finally, the small memory accesses that are performed by software (e.g. when loading a 32-bit CPU register) are merged if possible in order to use larger bursts in HW that produce less overhead and thus offer higher bandwidth. The resulting list of memory accesses can then be written to an IP core that is included with the workflow. When instantiated in an FPGA-SoC, this IP core

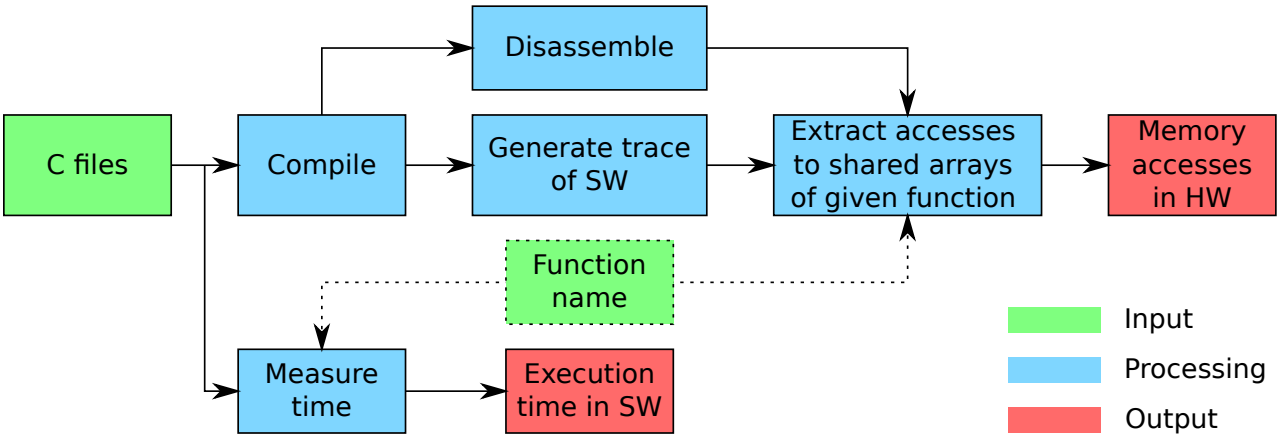

Figure 1: The presented workflow. Note that the IP core used to perform the memory accesses in HW is not shown.

will perform all the memory accesses and thus simulate an actual hardware implementation of the given C function. By measuring the number of cycles the memory accesses take and by summing up the amount of data of all the memory accesses in this function, the bandwidth in bytes/s can be calculated. For a comparison of the estimated bandwidth in hardware with the bandwidth of the SW implementation, the execution time of the given C function and therefore the bandwidth required by the SW can be measured.

Using the measured SW bandwidth and the estimated HW bandwidth, the user can make the decision on whether to move the function to HW or to stay with the SW implementation. The workflow can also be used to identify those functions that are suited best to be moved to HW in terms of memory bandwidth by executing the workflow for different functions.

3 Evaluation

In order to check the plausibility of the presented workflow, some micro-benchmarks have been evaluated: *FIR* (a 5-tap FIR filter), *intSqrt* (an iterative method to calculate square roots with integer precision using nested intervals), *memcpy* (a standard method for copying data between arrays) and *extract* (an image processing function that extracts a block of 5x5 samples out of a frame of 100x100 samples). The first three micro-benchmarks are performed on an array containing 100x32-bit integer values. The setup is as follows: For the SW results, an Intel i7-4770 CPU has been used with Linux 3.8. The HW results have been obtained using a Zynq-7045 from Xilinx with a single 64-bit port running at 187.5 MHz for memory access. As can be seen in Figure 2, *memcpy* requires the highest bandwidth. This matches the expectations as it consists mainly of coarse-grained memory accesses and performs almost no computations. *Extract* and *FIR* require lower bandwidths as they employ more fine-grained memory accesses (*extract*) or more computations (*FIR*). Finally, *intSqrt* requires the least bandwidth as it is computationally very expensive.

As the computational part of a HW implementation can often be performed very efficiently and fast, the memory accesses dominate over the computational part. Furthermore, due to the assumed BRAM buffers, the access patterns of *memcpy*, *FIR* and *intSqrt* implementations in hardware are very similar. As a result, they achieve a very similar bandwidth. Still,

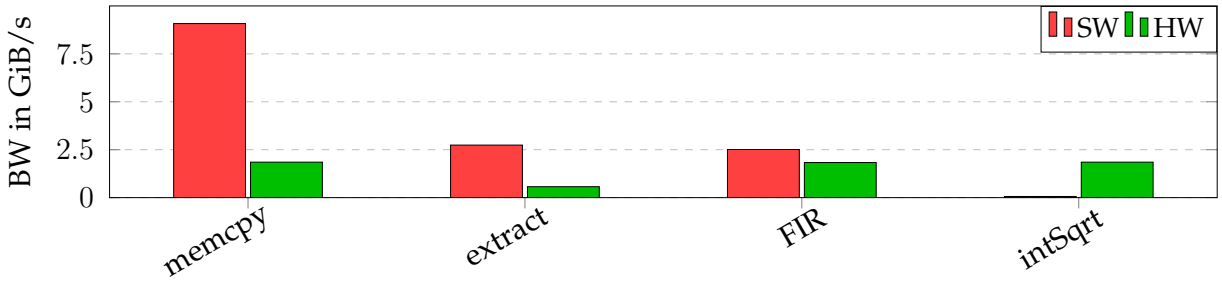

Figure 2: The results of the evaluation of the micro-benchmarks, i.e. the measured SW bandwidth and the estimated HW bandwidth using the presented workflow.

the memory bandwidth for *extract* is smaller due to more fine-grained memory accesses.

These results show that a HW/SW-Codesign of *intSqrt* is a reasonable idea. On the other hand, moving *memcpy* to HW is not reasonable at all as the caches of the i7 offer a significantly higher bandwidth for small amounts of data than an FPGA-SoC. As these results match the expectations, they indicate that the approach is plausible.

4 Conclusions and Future Work

In this paper, a trace-based workflow for evaluating application-specific memory bandwidth for FPGA-SoCs has been presented. By making two assumptions, a method was proposed that could estimate the memory bandwidth of a HW implementation by analyzing an equivalent SW implementation. An evaluation using several micro-benchmarks showed that the approach is reasonable and plausible. Future work includes a more thorough evaluation using a higher number of more complex benchmarks. Furthermore, the estimations of the HW bandwidth have to be verified using actual HW implementations of the benchmarks. Finally, an extension of the workflow to support evaluating more than one C function and to support various HW configurations will be performed.

References

[GCAMJ15] M. Göbel, C. Chi, M. Alvarez-Mesa, and B. Juurlink. High Performance Memory Accesses on FPGA-SoCs: A Quantitative Analysis. *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM 2015), Vancouver, Canada*, 2015.

[SSSS15] V. Sklyarov, I. Skliarova, J. Silva, and A. Sudnitson. Analysis and Comparison of Attainable Hardware Acceleration in All Programmable Systems-on-Chip. *2015 Euromicro Conference on Digital System Design (DSD), Funchal, Portugal*, 2015.

[SWWB13] M. Sadri, C. Weis, N. Wehn, and L. Benini. Energy and Performance Exploration of Accelerator Coherency Port using Xilinx ZYNQ. *ACM 10th FPGAWorld Conference, Copenhagen/Stockholm, Denmark/Sweden*, 2013.