

Göbel, M.; Elhossini, A.; Juurlink, B.

# A Methodology for Predicting Application-Specific Achievable Memory Bandwidth for HW/SW-Codesign

Conference paper | Accepted manuscript (Postprint)

This version is available at <https://doi.org/10.14279/depositononce-7079>



Göbel, M., Elhossini, A., & Juurlink, B. (2017). A Methodology for Predicting Application-Specific Achievable Memory Bandwidth for HW/SW-Codesign. In 2017 Euromicro Conference on Digital System Design (DSD). IEEE. <https://doi.org/10.1109/dsd.2017.40>

## Terms of Use

© © 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

WISSEN IM ZENTRUM  
UNIVERSITÄTSBIBLIOTHEK

Technische  
Universität  
Berlin

# A Methodology for Predicting Application-specific Achievable Memory Bandwidth for HW/SW-Codesign

Matthias Göbel, Ahmed Elhossini and Ben Juurlink  
Embedded Systems Architecture Group  
Technische Universität Berlin  
Berlin, Germany

Email: {m.goebel, ahmed.elhossini, b.juurlink}@tu-berlin.de

**Abstract**—The trend of using heterogeneous computing and HW/SW-Codesign approaches allows increasing performance significantly while reducing power consumption. One of the main challenges when combining multiple processing devices is the communication, as an inefficient communication configuration can pose a bottleneck to the overall system performance. To address this problem, we present a methodology that assists the designer in making good design decisions for systems using shared DDR memory for communication. Our methodology analyzes a software implementation of the application and subsequently predicts the memory accesses of a functionally equivalent hardware implementation of the selected function. We furthermore propose an IP core that can perform these predicted memory accesses to estimate the achievable memory bandwidth between a functionally equivalent hardware implementation and shared memory. The resulting achievable memory bandwidth estimations differ by less than 2% from the actual achievable memory bandwidth of a functionally equivalent hardware implementation, demonstrating the feasibility of the presented methodology.

**Keywords**—Memory access; FPGA-SoC; Memory bandwidth; HW/SW-Codesign; Memory estimation; Memory prediction

## I. INTRODUCTION AND RELATED WORK

HW/SW-Codesign approaches consisting of a processor and an FPGA can often be found in embedded systems today. In order to achieve maximum performance, it is necessary to not only optimize their software and hardware parts, but also to keep the shared memory bandwidth in mind. As processor and FPGA often use it for communication, it can pose a bottleneck to the overall system. Modern FPGA systems usually use external DDR memory for this task (e.g. FPGA-SoCs like Xilinx's Zynq-7000). The latency of a memory transaction can vary significantly, depending on the requested address and length as well as previous memory transactions. Therefore, it is almost impossible to estimate the available memory bandwidth for a specific application without a thorough analysis.

We present a novel approach to encounter the challenge of memory bandwidth estimation. Our approach is based on deriving information about a hardware component's memory access behavior from a software implementation offering the same functionality. In most cases, such software implementation is already available, either because an existing soft-

ware project should be accelerated using HW/SW-Codesign or because a software implementation has been written for verification of the hardware components to be implemented. The derived information can be used to approximate the memory access behavior of a potential hardware component before implementation. We also present an IP core that can use the gathered information to actually simulate the memory access behavior of the potential hardware component on an FPGA and measure the available application-specific memory bandwidth. By comparing the available memory bandwidth with the memory bandwidth of a software solution, the designer can make the decision of whether to follow a HW/SW-Codesign approach before starting the actual implementation.

Various work is available that discusses the memory bottleneck on FPGA systems [1], [2]. With the emerge of FPGA-SoCs, these devices have been increasingly used as a platform for HW/SW-Codesign. Multiple authors analyzed modern FPGA-SoCs and focused on measuring memory bandwidth for accesses from software as well as hardware under various scenarios [3]–[5]. Using their results, it is possible to roughly estimate the available memory bandwidth for a specific hardware implementation by comparing its memory access behavior to one of the evaluated scenarios. However, this requires a detailed knowledge about the memory access behavior of the specific application. Furthermore, as the memory bandwidth of DDR memory depends strongly on previous memory accesses (e.g. due to row precharging), the value of these studies is limited. Finally, *High-Level Synthesis* (HLS) tools [6] can be used to rapidly prototype a design. However, as the resulting HDL models are often less powerful and efficient than those involving handwritten HDL code, they do not actually solve the problem of memory bandwidth estimation for HW/SW-Codesign approaches.

The paper is organized as follows: Section II describes the involved challenges and our methodology to solve these. Furthermore, we present a workflow that implements our methodology. Section III discusses the models that are used by our approach to simulate the memory access behavior of an actual design following a HW/SW-Codesign approach. In Section IV, our implemented workflow is applied to different

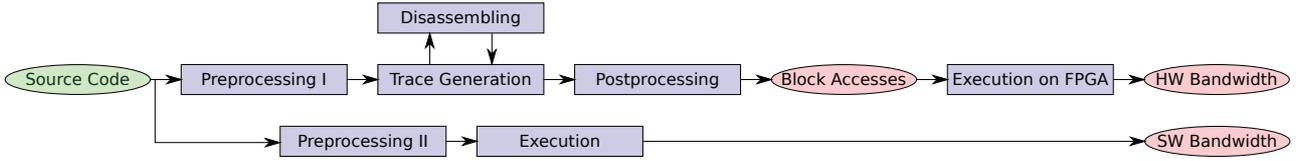


Figure 1: An overview of the workflow for estimating the hardware memory bandwidth and measuring the software memory bandwidth for a given function.

use cases and the results are analyzed. Section V concludes the paper and presents future work.

## II. METHODOLOGY AND WORKFLOW

Estimating the potential memory bandwidth of a hardware implementation by analyzing a software implementation poses several challenges. First of all, the memory access behavior of both implementations will differ a lot. As processor cores typically do not contain large memory buffers but only small registers, data processing in software is usually based on the idea of loading small amounts of data into a processor, processing them and storing the results back to memory. Small memory accesses to DDR memory, however, are expensive in terms of latency and therefore limit the overall performance. Hardware implementations, on the other hand, often contain memory buffers that allow storing large amounts of data locally. Therefore, larger memory accesses can be performed between DDR memory and these buffers than between DDR memory and a processor, thus increasing the potential overall performance.

In the following we present our methodology by describing an actual workflow that implements our methodology based on this discussion. An overview of the workflow can be seen in Figure 1. It accepts the source code of a software implementation of the application as an input and provides the memory bandwidth of the software implementation of a given function as well as the estimated memory bandwidth of a hardware implementation of the same function. By comparing these results to each other, a well-founded decision can be made whether to follow a HW/SW-Codesign approach. The designer has to specify that function in the software implementation that should be implemented in hardware as well as those data structures that are shared between this function and the rest of the application.

The estimation of the hardware memory bandwidth basically consists of three steps:

- Generating a memory trace of the function that should be moved to hardware
- Converting this trace to an equivalent trace of a hardware implementation of the same function
- Executing this trace on an FPGA and measuring the available memory bandwidth

These steps will be discussed in the following.

### A. Generating memory trace

The estimation of the hardware bandwidth starts with the preprocessing of the source code. This involves inserting instructions into the source code that gather information about those data structures that should be shared between hardware and software components. In particular, the addresses and sizes of all such data structures are gathered. The identification of such data structures must be performed by the designer as it poses a major design decision.

After the code preprocessing is complete, the actual memory trace of the application is generated. For this task, we use the Valgrind instrumentation framework [7].

Once this full trace has been generated, it is necessary to extract only those memory accesses that belong to the function that should actually be moved to hardware. This task can be performed by disassembling the compiler-generated executable and comparing the resulting assembly code with the full trace: Only those load and store operations that belong to instructions from the specified function must be analyzed. Also, all instruction fetch accesses can be discarded because a hardware implementation with the same functionality would not fetch any instructions from memory.

### B. Converting the memory trace to an equivalent hardware memory trace

After these memory accesses have been extracted, multiple postprocessing steps are required in order to get those memory accesses that would be performed by a corresponding hardware implementation as described previously. First of all, only memory accesses to data structures shared between software and hardware must be considered. Accesses to internal data structures of the selected function can be omitted as such structures can be implemented as local memory inside a hardware implementation of the function.

Furthermore, the granularity of the memory accesses has to be changed: As discussed before, while the memory accesses of a software implementation will be very fine-grained, a hardware implementation can buffer large amounts of data in local buffers and therefore use larger, burst-capable memory accesses. Finally, the virtual addresses of the memory trace have to be converted into physical addresses and long data accesses have to be split to obey 4K boundaries and the maximum size of bursts supported by the memory controller.

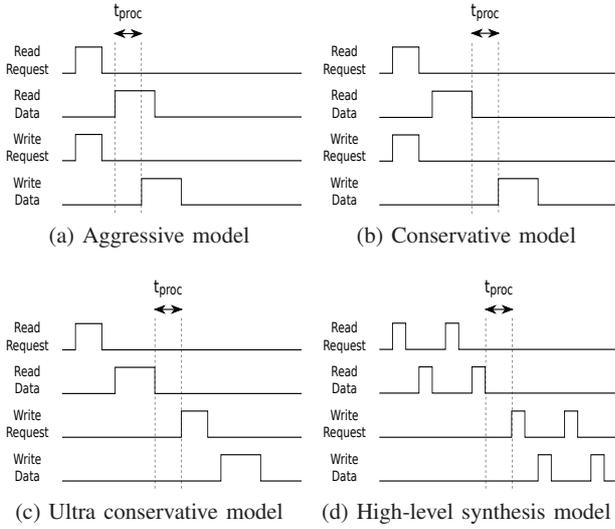


Figure 2: The timing behavior of the models used in the presented approach with  $t_{proc}$  describing the processing latency.

### C. Performing memory accesses on FPGA and measuring bandwidth

The resulting list of memory accesses can be used to simulate the memory access behavior of a hardware implementation of the specified function. An IP core has been designed that contains a buffer that stores the trace as well as all the necessary functionality to actually perform these memory accesses and measure the resulting bandwidth.

### D. Measuring the memory bandwidth of the software implementation

The estimated memory bandwidth for a hardware component can only be assessed in relation to a baseline. For this reason, the presented workflow also measures the memory bandwidth for the original software implementation. An overview of the required steps can also be seen in Figure 1. The preprocessing stage (*Preprocessing II* in the figure) involves adding time measurement functionality to the original source code. Using the elapsed time and the total amount of data that is stored or loaded to/from main memory, the resulting memory bandwidth can be calculated.

## III. TIMING BEHAVIOR OF MEMORY TRANSACTIONS

The list of memory accesses that is generated when using the workflow and written to the aforementioned buffer in the IP core does not contain any timestamps or other information about the timing behavior of the memory accesses. However, the order in which the memory accesses are executed is not negligible as the latency of a DDR memory access depends on previous accesses: Two consecutive accesses to the same

row in DDR memory will result in a higher bandwidth than two consecutive accesses to different rows.

The following assumptions are employed in the presented work flow:

- All data to be written depends on some previously read data, i.e. writing data cannot start before data has been read
- Data can be stored inside the hardware component of the design in a temporary buffer consisting of block RAMs

As these assumptions are rather general, they hold for a large range of applications.

Based on these assumptions, we defined four *models* for the timing behavior of the HW memory accesses. They are illustrated in Figure 2. As current FPGA-SoCs mainly use the AXI protocol, they are designed for such busses. However, similar busses can be employed as well. The designer has to pick one model that suites his application best. The **Aggressive** model assumes that all read and write requests can already be issued at the beginning, i.e. before reading or writing any data. In order for this to work, all addresses and access lengths must be static and therefore independent of the data. It also assumes that reading and writing data can overlap, i.e. a single result does not depend on all input samples. A huge variety of typical data processing steps can be implemented in such a way, e.g. one-dimensional FIR filters. The **Conservative** model also assumes that all requests can be issued at the beginning. However, in this model the writing of the results only starts after all input samples have been read. Therefore, the results can depend on all input samples. An example for such a behavior would be two-dimensional FIR filters which first process a block of data horizontally and afterwards vertically. The **Ultra Conservative** model makes the same assumptions regarding data dependency as the conservative model. However, in this model the write requests are not issued until all data has been read thus allowing the write request to depend on the the input data. Compression algorithms like Huffman Coding can be considered a use-case for this model. The **High-level Synthesis** model resembles the timing behavior of IP cores generated with Xilinx's *Vivado HLS* suite. While being similar to the Ultra Conservative model, it issues a single request and then waits until the required data has been read or written before issuing the next request.

Besides choosing one of these general models, the designer also has to define four model parameters. The definition of the **Processing Latency** depends on the selected model and can be seen in Figure 2 as  $t_{proc}$ . It describes the latency that is required to actually process the input data and may depend on the number of input samples. The **Read and Write Widths**, i.e. the maximum numbers of input samples that can be read or written in one cycle depends on the sample depth and the width of the AXI bus. However, a hardware component might not provide a sufficient data

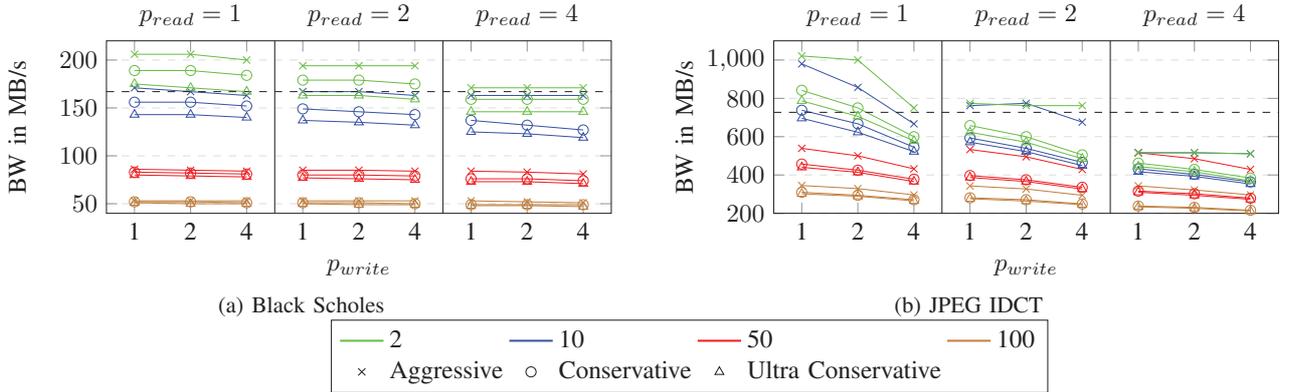


Figure 3: The results for Black Scholes and JPEG IDCT with the colors indicating  $t_{proc}$  and the marks indicating the selected model. The required software bandwidths according to our workflow (visualized as a dashed line) are 167 (Black Scholes) and 727 (JPEG IDCT) MB/s, respectively. Note that the bandwidth axis does not start at 0 for clarity.

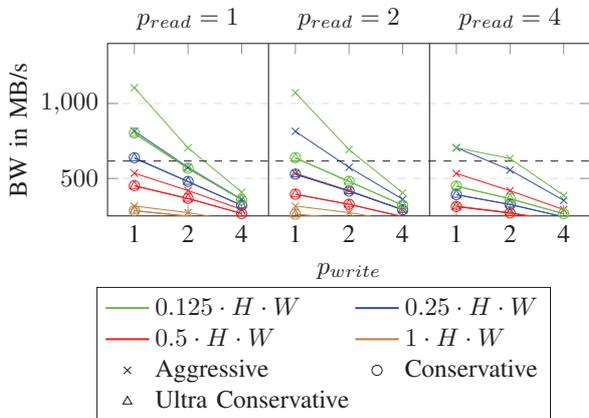


Figure 4: The results for HEVC Motion Compensation with the colors indicating  $t_{proc}$  and the marks indicating the selected model. The required software bandwidth according to our workflow (visualized as a dashed line) is 616 MB/s. Note that the bandwidth axis does not start at 0 for clarity.

processing throughput to utilize the full memory bandwidth. Therefore, the designer has to provide parameters  $p_{read}$  and  $p_{write}$ . For example, a value of  $p_{read} = 4$  can be interpreted as reading data only every fourth cycle. Finally, for the HLS model the designer also has to provide the **Data-to-Request latency**, i.e. the latency between reading/writing data and issuing the next read/write request.

#### IV. EVALUATION

In this section, an implementation of the presented workflow is first applied to three different data processing algorithms to show how to actually use the workflow. Afterwards, the bandwidth estimations are verified by actually

implementing them following a HW/SW-Codesign approach and measuring their memory bandwidths.

##### A. Application of the workflow

The workflow has been applied to three different applications that are representatives for a huge variety of data processing algorithms:

- *Fast integer IDCT* of the JPEG reference decoder [8]
- *Black Scholes option pricing* from the PARSEC benchmark suite [9]
- *Horizontal motion compensation* of a commercial HEVC decoder [10]

For the evaluation, an Intel i7-4770 CPU has been used for running the workflow and a Xilinx Zynq-7045 for estimating the actual memory bandwidth for HW/SW-Codesign. For memory accesses, a single 64-bit High-Performance AXI port has been chosen. In order to evaluate the maximum bandwidths possible for the chosen parameters, the highest frequency possible with the Zynq have been used, i.e. 250 MHz. Different values have been chosen for the parameters discussed in Section III. Figures 3 and 4 show the results.

The processing latency  $t_{proc}$  has been chosen to be fixed for Black Scholes and JPEG IDCT as both work on fixed block sizes. On the other hand, the block size can vary from  $8 \cdot 4$  samples up to  $64 \cdot 64$  samples for motion compensation. Therefore, different degrees of parallelism depending on the block height and width have been evaluated. For the read and write widths, all possible combinations of one data beat every cycle, every second cycle and every fourth cycle has been chosen.

The results can be interpreted as follows:

- *Black Scholes*

$p_{read}$  and  $p_{write}$  have only a minor influence on the bandwidth for this application. This suggests that the

	Black Scholes	JPEG IDCT	HEVC
$t_{proc}$	425	159	$4 \cdot (W - 7) \cdot H + 4 \cdot (H - 1) + 121$
$p_{read}$	1	8	1
$p_{write}$	1	1	4
Measurement (MB/s)	15.12	180.45	29.18
Estimation (MB/s)	15.09	177.78	29.00
Error (%)	0.2	1.5	0.6

Table I: An overview of the verification process

application is not memory-bound for most configurations. However,  $t_{proc}$  must be less than 10 cycles and the aggressive model should be chosen, if possible, to achieve the bandwidth of the software implementation.

- **JPEG IDCT**

In this case,  $p_{read}$  and  $p_{write}$  influence the bandwidth significantly. In fact,  $p_{read}$  must be lower than 4 to be able to achieve the bandwidth of the software implementation. Furthermore,  $t_{proc}$  should be less than 50 cycles. For  $p_{read} = 1$  and  $t_{proc} = 2$ , all three models can achieve the required bandwidth.

- **HEVC Motion Compensation**

The estimations show that low values for  $p_{read}$  and  $p_{write}$ , i.e. an efficient memory interface inside the hardware implementation, are important. Furthermore, the estimations indicate that a degree of parallelism of 4 ( $0.25 \cdot H \cdot W$ ) or 8 ( $0.125 \cdot H \cdot W$ ) and the aggressive model should be used to meet the required bandwidth.

### B. Verification of the methodology

In order to verify the accuracy of the estimated bandwidths, all three use-cases have been implemented using Vivado HLS. The designs have been analyzed to derive the parameters for the workflow and the memory bandwidths have been estimated using the *HLS* mode (see Section III). Furthermore, the memory bandwidths of the actual design have been measured. The results are presented in Table I. As the estimation error is rather small (less than 2% for all three applications), it can be concluded that the proposed methodology is feasible and the workflow can indeed be used to estimate the achievable memory bandwidth of a design following a HW/SW-Codesign approach before actually starting the implementation.

## V. CONCLUSIONS AND FUTURE WORK

We presented a novel approach for estimating the available memory bandwidth for hardware components used in a HW/SW-Codesign approach. It analyzes a software implementation with the same functionality, generates a memory trace of this implementation when processing real workloads and creates a list of corresponding memory accesses that would be performed by a hardware implementation. A provided IP core can be instantiated on a target device, e.g. an FPGA-SoC, and perform these memory accesses to

finally estimate the available memory bandwidth. In order to support a broad range of applications, different models for memory access behavior have been presented that can be used to match the actual behavior of the application to be analyzed. An evaluation has been performed that presents the use of the workflow implementing our approach for different representative applications. Furthermore, the estimations have been compared to an HLS implementation of these applications for verification. It shows that the estimation error is less than 2% for all applications, thus indicating the feasibility of our approach.

A sufficient memory bandwidth is only one aspect that has to be considered in a design space exploration. Other aspects include device utilization and power consumption as well as the throughput inside a hardware component. Future work could include an analysis of such aspects to allow for a full design space exploration for HW/SW-Codesign.

## REFERENCES

- [1] H. Fu and R. Clapp, *Eliminating the Memory Bottleneck: An FPGA-based Solution for 3D Reverse Time Migration*, 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, USA, 2011.
- [2] Y. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman and P. Wei, *A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms*, 53rd ACM/EDA/IEEE Design Automation Conference (DAC), Austin, USA, 2016.
- [3] M. Göbel, A. Elhossini, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *A Quantitative Analysis of the Memory Architecture of FPGA-SoCs*, 13th International Symposium on Applied Reconfigurable Computing (ARC), Delft, Netherlands, 2017.
- [4] M. Sadri, C. Weis, N. Wehn and L. Benini, *Energy and Performance Exploration of Accelerator Coherency Port Using Xilinx ZYNQ*, 10th ACM FPGAWorld Conference, Copenhagen/Stockholm, Denmark/Sweden, 2013.
- [5] 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.
- [6] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson and K. Bertels, *A Survey and Evaluation of FPGA High-Level Synthesis Tools*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 35, Issue 10, 1591-1604, 2015.
- [7] N. Nethercote and J. Seward, *Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation*, 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), San Diego, USA, 2007.
- [8] *Independent JPEG Group*, <http://www.ijg.org/>
- [9] C. Bienia, S. Kumar, J. Singh and K. Li, *The PARSEC Benchmark Suite: Characterization and Architectural Implications*, 17th International Conference on Parallel Architectures and Compilation Techniques (PACT), Toronto, Canada, 2008.
- [10] *Spin Digital Video Technologies*, <http://www.spin-digital.com>