in Proceedings of the 12th International Symposium on Applied
Reconfigurable Computing (ARC), Rio de Janeiro, Brazil, 22-24 March, 2016.

A *Lost Cycles* Analysis for Performance Prediction using High-Level Synthesis

Bruno da Silva, Jan Lemeire, An Braeken, and Abdellah Touhafi

Vrije Universiteit Brussel (VUB),
INDI and ETRO department, Brussels, Belgium
{bruno.da.silva,jan.lemeire,an.braeken,abdellah.touhafi}@vub.ac.be

Abstract. Today's High-Level Synthesis (HLS) tools significantly reduce the development time and offer a fast design-space exploration of compute intensive applications. The difficulty, however, to properly select the HLS optimizations leading to a high-performance design implementation drastically increases with the complexity of the application. In this paper we propose as extension for HLS tools a performance prediction for compute intensive applications consisting of multiple loops. We affirm that accurate performance predictions can be obtained by identifying and estimating all overheads instead of directly modelling the overall execution time. Such performance prediction is based on a cycle analysis and modelling of the overheads using the current HLS tools' features. As proof of concept, our analysis uses Vivado HLS to predict the performance of a single-floating point matrix multiplication. The accuracy of the results demonstrates the potential of such kind of analysis.

Keywords: High-Level Synthesis, Lost Cycles, FPGA, Performance Prediction, Overhead analysis

1 Introduction

High-Level Synthesis (HLS) tools allow FPGA designers to develop their implementations in high level languages such as C or C++. Despite the high-level representation of the algorithms accelerates the Design-Space Exploration (DSE), the multiple implementation choices, thanks to the large set of available optimizations, make the DSE a non-trivial task. Since this kind of tools provide detailed information about the latency, frequency and an estimation of the resource consumption of each implementation, we believe that current HLS tools manage enough information to provide an accurate performance prediction. Such performance prediction would lead to a much faster DSE of compute intensive applications.

To address this challenge, we propose a cycle-based analysis for performance prediction as extension for HLS tools. The principles of such analysis are originally presented in [1], [2], where the overheads of parallel programs are identified and modelled in order to predict performance. The performance of any implementation reflects not only the quality of the implementation but also the impact of its overheads. We propose an in-depth performance analysis based on those inefficiencies. The identification, the classification, and the modelling of those overheads would lead to an accurate performance prediction. The analysis of the inefficiencies, referred as overheads from now on, drives to a model which can be used to guide and to reduce the DSE. Consequently, the most appropriated HLS optimization can be used to reduce a particular overhead, increasing the overall performance. Our purpose is to demonstrate that a *lost cycles* analysis of compute intensive kernels leads to an accurate performance prediction using HLS tools.

This paper is organized as follows. Section 2 presents related work. The definition of *lost cycles* and their classification is introduced in Section 3. A methodology using the proposed analysis is detailed in Section 4. In Section 5 our *lost cycles* analysis is applied to a well-known algorithm. Finally, the conclusions are drawn in Section 6.

2 Related work

During the last years, researchers have proposed many techniques to reduce the DSE. We focus on some of those models which predict and model performance based on the exploration of the available HLS' optimizations. Several papers such as [3] incorporate certain overhead predictions in their performance models. Their overhead model, however, only targets loop unrolling and do not consider the use of HLS tools. The authors in [4] reduce the DSE by first modelling the performance and the area before evaluating their prediction using Vivado HLS. However, their approach only target nested loops and loop unrolling while we consider different loop hierarchies and multiple optimizations. In [5] a *divide and conquer* algorithm to accelerate the DSE using HLS tools is presented. They propose a partition of the algorithm for their individual and exhaustive design exploration by invoking the HLS tool, which leads to a long simulation and synthesis runtime. Our approach only requires a short DSE using an HLS tool to elaborate accurate performance predictions. Finally, the authors in [6] use several HLS tools for their performance prediction. However, this approach does not predict the performance of a single design neither the potential impact of the optimizations, which can be obtained thanks to the *lost cycles* analysis.

Most of the strategies elaborate their performance models based on the performance analysis of multiple configurations. Our approach targets the modelling of the overheads in order to generate a more accurate and less time-consuming DSE. As far as we know, this is the first time a *lost cycles* analysis is applied for FPGAs, specially considering HLS tools.

3 Definition of *Lost Cycles*

A clock cycle can be considered as the minimum unit of time that one operation needs to be executed. The clock cycle is defined by the operational frequency of the design. Thus, despite the number of cycles to execute an operation changes based on the target frequency, the time for one operation remains constant. Our cycle-based analysis considers the frequency, which defines the accuracy of the use of clock cycles as time unit, when calculating the performance.

Let the useful operations be the main operations characterizing the algorithm to be implemented. The *lost cycles* (L_o) of a design are all those clock cycles which are not dedicated to compute useful operations. The remaining clock cycles are the useful cycles, also called *compute cycles* (L_c), which are the cycles dedicated to compute useful operations. The overall number of clock cycles (L_{imp}) that an algorithm composed by n loops $\{L_1, L_2, ...L_n\}$ needs can be expressed as Eq. 1.

$$L_{imp} \;=\; [((L_{c_1} + L_{o_1}) \cdot I_{L_1} + (L_{c_2} + L_{o_2})) \cdot I_{L_2} + ... \;+ L_{c_n} + L_{o_n}] \cdot I_{L_n} \quad (1)$$

Eq. 1 reflects the loop hierarchy by grouping those cycles associated to each loop. L_{c_i} and L_{o_i} are the useful and lost cycles associated to loop i respectively, while I_{L_i} is the number of iterations of loop i. This equation can be rearranged to group all the useful and lost cycles as shown in Eq. 2.

$$\begin{aligned} L_{imp} = {} & [(L_{c_1} \cdot I_{L_1} + L_{c_2}) \cdot I_{L_2} + ... + L_{c_n}] \cdot I_{L_n} \\ & + [(L_{o_1} \cdot I_{L_1} + L_{o_2}) \cdot I_{L_2} + ... + L_{o_n}] \cdot I_{L_n} \\ & = L_c + L_o \end{aligned} \quad (2)$$

The use of the standalone latency reported by the HLS tool as metric of the design performance could lead to wrong conclusions. In order to avoid any incoherence, we use the latency reported at certain frequency (F_{imp}) and the assumption that L_c is the minimum number of clock cycles to compute at such clock rate (Clk_{imp}). Let OP_{imp} be the number of operations executed in L_{imp}, the execution time of a design (t_{imp}) can be expressed as $t_{imp} \;=\; Clk_{imp} \cdot L_{imp}$. Hence, Eq. 3 defines the design performance (P_{imp}) and, based on Eq. 2, reflects how the overheads affect.

$$P_{imp} \;=\; \frac{OP_{imp}}{t_{imp}} \;=\; \frac{OP_{imp}}{Clk_{imp} \cdot (L_c + L_o)} \quad (3)$$

3.1 Assumptions

Before development of our overhead analysis, we make a few more assumptions. Firstly, we assume that each operation is exclusively composed by dedicated amount of resources. Therefore, the consumed resources are not reused to execute different operations. Secondly, despite different types of operations (integer, floating point, ...) must be considered, we only consider single floating-point operations (FP) to introduce our methodology. Thirdly, to facilitate the explanation of our definitions, only DSPs are considered despite FP operations can also be implemented using logic resources. This assumption, for instance, simplifies the equation to obtain L_c. Nevertheless, L_c can be obtained for other type of operations, such as fixed-point integer operations, thanks to the reports of current HLS tools. For the sake of simplicity, our analysis targets application kernels consisting of multiple loops and compute intensive operations.

3.2 Identifying the Useful Cycles

The number of useful cycles is directly related to the number of operations implemented on the FPGA. Therefore, it is possible to obtain L_c from the resource consumption of a particular implementation. Let be OP_p the number of operations which can be executed in parallel and L_{DSP} the minimum latency of a FP operation implemented exclusively using DSPs. Eq. 4 relates L_c with the consumed area and shows how L_c decreases when the number of operations executed in parallel increases.

$$L_c \; = \; \frac{L_{DSP} \cdot OP_{imp}}{OP_p} \tag{4}$$

3.3 The *Lost Cycles* Classification

The proper selection of *lost cycles* categories is needed for our approach. We propose the following categories:

Initialization Overhead (O_{Init}) The initialization overhead models those overheads related to the filling of the stages of a pipelined operation with streaming data.
Non-overlapping memory accesses (O_{Mem}) The non-overlapping memory accesses are all those memory accesses which are not overlapped in time with any useful computation or with another latency overhead.
Logic control (O_{Ctrl}) The latency related to the loop control or to the synchronization of the operations belong to the logic control category of overhead.

$$L_o \; = \; O_{Init} \; + \; O_{Mem} \; + \; O_{Ctrl} \tag{5}$$

Those categories are similar, but not the same as the latency overheads identified in [3]. It may occurs that, due to pipeline operations, two overheads are overlapped. Those overlapped *lost cycles*, however, must be considered only once and included in one of the categories.

4 Proposed Methodology Overview

The proposed methodology consists in a *lost cycle* analysis using the HLS tool while exploring the target optimizations. During an initial exploration phase the impact of each optimization on the HLS design is profiled in order to fetch our *lost cycle* analysis. Figure 1 shows how this initial DSE helps to identify, classify and quantify each *lost cycle* in order to generate a model for each overhead. The analysis of L_o is only possible when L_c is obtained since L_{imp} is already reported by the HLS tool. The parameters in Eq. 4 used to calculate L_c can be extracted from the reports of the HLS tool, from the hardware specifications or from empirical study.

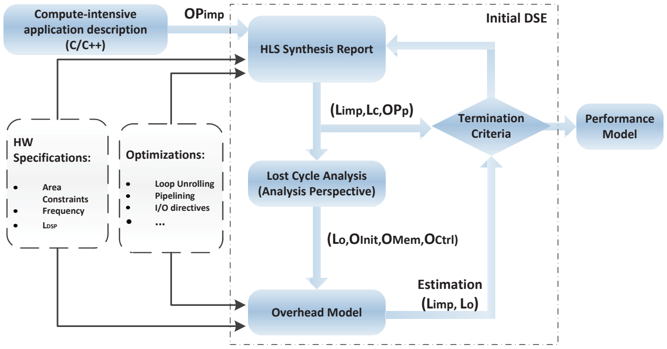

Fig. 1: *Main Steps of the proposed methodology.*

- OP_{imp} represents the total number of operations which must be executed and is usually known.
- L_{DSP} is specified by the hardware and is usually available in technical documents [7]. For instance, the reported latency for FP additions with our target FPGA is 1, meaning that one FP operation can be computed per clock cycle.
- OP_p determines the level of concurrency and is extracted from the resource consumption. OP_p can be easily obtained using current HLS tools, where the resource consumption is reported for each particular design.

Once L_c, and consequently L_o are obtained for a particular design, the overhead analysis can continue exploring the latency impact of the application size, the compiler optimizations or source code modifications. When each overhead is properly modelled, L_o can be estimated following Eq. 5. The performance prediction of a configuration (optimizations, hardware specifications,...) is obtained from the L_o estimation, which leads to L_{imp}, since L_c remains constant while none of the parameters of Eq. 4 changes.

Our methodology is designed to exploit the Vivado HLS features. Vivado HLS not only generates a synthesis report for every design solution but also provides a useful *analysis perspective* which details the intermediate operations. Both, the information reported by the compiler and the information extracted from the *analysis perspective* are used to fetch our analysis. For instance, Figure 2 exemplifies what information can be extracted from *Analysis perspective* for a FP matrix addition.

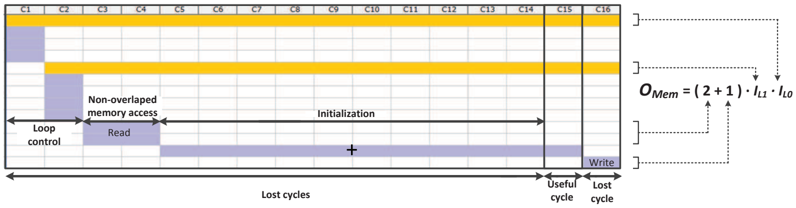

Fig. 2: *Example of how the overheads of a FP matrix addition can be modelled.*

Table 1: *Example of a short DSE to profile the the optimizations and to generate a model of each overhead.*

Matrix Size	**FLOPs** (OP_{imp})	**Optimizations**	**Latency** (L_{imp})	O_{Init}	O_{Mem}	O_{Ctrl}
4x4	128	None	1066	768	128	106
		PLU x2 L0	1064	768	128	104
		PLU x2 L1	1058	768	128	98
		PLU x2 L2	858	608	96	90
		Pipeline L0	97	27	3	3
		Pipeline L1	103	33	3	3
		Pipeline L2	706	528	32	3
32x32	65536	None	526402	393216	65536	34882
		PLU x2 L0	526386	393216	65536	34866
		PLU x2 L1	525890	393216	65536	34370
		PLU x2 L2	412738	311296	49152	19522
		Pipeline L0	32786	12	3	3
		Pipeline L1	33010	229	3	10
		Pipeline L2	274434	234496	2048	5122

HLS synthesis report: Vivado HLS generates at every compilation a detailed report with useful information about latency and resource consumption. It's possible to derive from Eq.4 that L_c equals Op_{imp} since each FP addition is implemented exclusively with DSPs, and both OP_p and L_{DSP} are one [7]. The iteration time together with the number of iterations determines the number of clock cycles needed by the implementation.

Lost cycle analysis: The execution trace depicted in Figure 2 shows how the *Analysis perspective* allows to identify and quantify the different overheads in great detail. For instance, one FP addition requires 11 clock cycles to be completed but only 1 clock cycle can be considered useful. Extra cycles like O_{Init} are consumed by the inner operations needed to compute any FP operation. O_{Mem} includes two clock cycles to load the input values and one extra cycle to write the output. Finally, O_{Ctrl} is needed to initiate the iteration of the loop and to check the exit condition.

Overhead model and performance estimation: Once O_{Init}, O_{Mem} and O_{Ctrl} have been measured they can be modelled based on the selected configuration of the design. Their modelling is possible by analysing the variation of the parameters of the target configuration. The impact of the optimizations, the source code modifications or the application scaling on L_o differs.

Performance estimation: Once the *lost cycles* are properly modelled, the performance prediction for one implementation can be easily calculated (Eq.3).

The main target is the elaboration of a table such as Table 2, where the overheads are modelled based on the explored optimizations.

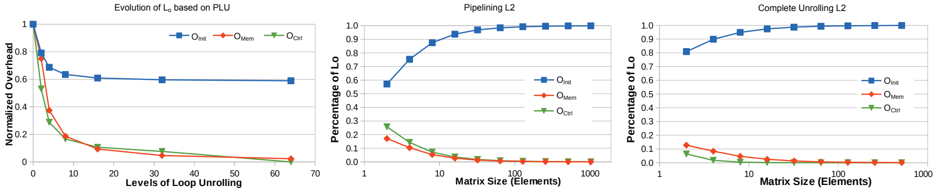

Fig. 3: *Modelling the impact of L_o when using different optimizations.*

Table 2: *Summary of the overhead models based on the optimizations explored.*

Opt.	**Loop**	O_{Init}	O_{Mem}	O_{Ctrl}
None	–	$(7+5) \cdot I_{L0} \cdot I_{L1} \cdot I_{L2}$	$2 \cdot I_{L0} \cdot I_{L1} \cdot I_{L2}$	$(I_{L1} \cdot (I_{L2}+2)+2) \cdot I_{L0}+2$
PLU	L0	$\frac{I_{L0}}{U_0} \cdot I_{L1} \cdot I_{L2} \cdot (U_0 \cdot (7+5))$	$2 \cdot U_0 \cdot \frac{I_{L0}}{U_0} \cdot I_{L1} \cdot I_{L2}$	$\frac{I_{L0}}{U_0} \cdot (U_0 \cdot I_{L1} \cdot (I_{L2}+2)+U_0+1)+2$
	L1	$I_{L0} \cdot \frac{I_{L1}}{U_1} \cdot I_{L2} \cdot (U_1 \cdot (7+5))$	$2 \cdot U_1 \cdot I_{L0} \cdot \frac{I_{L1}}{U_1} \cdot I_{L2}$	$I_{L0} \cdot (\frac{I_{L1}}{U_1} \cdot (U_1 \cdot I_{L2}+U_1+1)+2)+2$
	L2	$I_{L0} \cdot I_{L1} \cdot \frac{I_{L2}}{U_2} \cdot (U_2 \cdot 7+5)$	$3 \cdot I_{L0} \cdot I_{L1} \cdot \frac{I_{L2}}{U_2}$	$I_{L0} \cdot (I_{L1} \cdot (\frac{I_{L2}}{U_2}+3)+2)+2$
Pipeline	L0	$7+5$	3	3
	L1	$(7 \cdot I_{L0})+5$	3	$3+\frac{I_{L0}}{4}$
	L2	$(7+5)+7 \cdot I_{L0} \cdot I_{L1} \cdot (I_{L2}-1)$	$3 \cdot I_{L0} \cdot I_{L1}$	$2+2 \cdot (2 \cdot I_{L0}) \cdot I_{L1}$

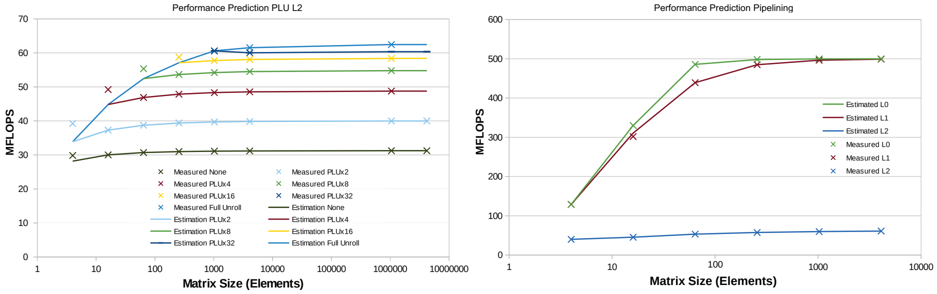

Fig. 4: *Comparison of our performance predictions versus HLS reports.*

5 Experimental Results

Despite we consider that our methodology can be potentially automated, many steps still need to be manually elaborated. The main purpose of this case study is to exemplify how our overhead analysis metric is elaborated using the information provided by an HLS tool in a reasonable time. Our DSE is done using Vivado HLS 2014.2 targeting an Xilinx Virtex6 lx240t FPGA at 250 MHz.

The implemented FP matrix multiplication consists of an outer-most loop (L0), a the middle loop (L1) and an inner-most loop (L2). No register is used in L2 to store intermediate accumulated values. Table 1 summarizes the Vivado HLS reports when different optimizations are applied. Vivado HLS reports the total number of clock cycles required to execute all the computations (L_{imp}). OP_p is extracted from the number of instances reported in the resource estimation. By default, Vivado HLS instantiates two dedicated cores to execute the addition and the multiplication. Op_{imp} represents the number of additions that must be computed, which is simply $2 \cdot I_{L0} \cdot I_{L1} \cdot I_{L2}$. Consequently, L_c is obtained from Eq. 4 and equals $I_{L0} \cdot I_{L1} \cdot I_{L2}$, which is the minimum number of clock cycles that a matrix multiplication needs when only 2 dedicated operations are executed in parallel. The addition and the multiplication require 7 and 5 clock cycles respectively to be initialized (O_{Init}). O_{Mem} only demands 2 clock cycles because the cycles dedicated to write the generated output back ($I_{L0} \cdot I_{L1}$) are overlapped with O_{Ctrl}. The modelling of each overhead is summarized in Table 2.

The left figure in Figure 3 depicts the evolution of the *lost cycles* of a matrix with 64 × 64 elements when loop L2 is unrolled. A higher impact is obtained for low levels of unrolling. Further levels of unrolling evidence that O_{Init} dominates L_o and can not be reduced beyond a certain limit. The middle and the right figures in Figure 3 show the evolution of the overheads while increasing the matrix size. In both cases, L_o is dominated by O_{Init}. This result is expected since O_{Init} is determined by the latency of the FP operations.

Figure 4 compares our performance predictions and the HLS reported performance. The left figure shows the impact of unrolling loop L2. Notice how our performance prediction is slightly pessimist when the loop is completely unrolled. The equations in Table 2 consider additional control, which increases O_{Ctrl} but that is removed when the loops are completely unrolled. Nevertheless, our predictions are extremely accurate for any level of unrolling or matrix size. The right figure shows the same comparison when pipelining loops, where our performance prediction achieves a high accurate estimation. Both figures depict how our technique performs an accurate prediction, which leads to a fast DSE.

6 Conclusion

The modelling of the overheads, a concept originally designed to analyse performance of parallel software, has been successfully adapted to the domain of FPGAs. Our proposed methodology shows an accurate performance prediction and enough flexibility to be applied for complex designs. Despite this approach is still manually elaborated, we believe that current HLS tools can be extended to offer such kind of performance prediction for the designer. Nevertheless, the automation of our methodology is our main priority as future work.

References

1. Crovella, M. E., et al. ”The search for lost cycles: A new approach to parallel program performance evaluation”, *Rochester University NY Dept of computer science*, 1993
2. Crovella, M. E., et al. ”Parallel performance prediction using lost cycles analysis”, *In Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, (pp. 600-609). IEEE Computer Society Press, 1994
3. Park, J., et al. ”Performance and Area Modeling of Complete FPGA Designs in the Presence of Loop Transformations”, *Computers*, IEEE Transactions on, 53(11), pp. 1420-1435. 2004
4. Zhong G., et al. ”Design Space Exploration of Multiple Loops on FPGAs using High Level Synthesis”, *In Computer Design (ICCD)*, 32nd IEEE International Conference on (pp. 456-463). IEEE. 2014
5. Schafer, B. C., et al. ”Divide and conquer high-level synthesis design space exploration”, *ACM Transactions on Design Automation of Electronic Systems* (TODAES), 17(3), 29. 2012
6. da Silva, B., et al. ”Performance modeling for FPGAs: extending the roofline model with high-level synthesis tools”, *International Journal of Reconfigurable Computing*, 7, 2013.
7. Xilinx. Xilinx logicore IP floating-point operator v6.1 product specification. Technical report, Xilinx, 2012.