Performance and Resource Modeling for FPGAs using High-Level Synthesis tools

Bruno DA SILVA [a], An BRAEKEN [a], Erik H. D'HOLLANDER [b] and Abdellah TOUHAFI [a]

[a] *Erasmus University College, IWT Dept., Brussels, Belgium*
[b] *Ghent University, ELIS Dept., Ghent, Belgium*

Abstract. High-performance computing with FPGAs is gaining momentum with the advent of sophisticated High-Level Synthesis (HLS) tools. The performance of a design is impacted by the input-output bandwidth, the code optimizations and the resource consumption, making the performance estimation a challenge. This paper proposes a performance model which extends the roofline model to take into account the resource consumption and the parameters used in the HLS tools. A strategy is developed which maximizes the performance and the resource utilization within the area of the FPGA. The model is used to optimize the design exploration of a class of window-based image processing application.

Keywords. Roofline Model, FPGA, High-Level Synthesis

Introduction

Field Programmable Gate Arrays (FPGAs), organized as a programmable and massively parallel architecture, offer a great performance potential. However, the design effort of FPGAs requires a detailed knowledge of hardware and a significant time consumption. New High-Level Synthesis (HLS) environments continue to improve the creation and the optimization of a design. The HLS tools reduce the development time and automate the compilation and synthesis flow from high-level languages, such as C/C++ or SystemC, to register transfer level languages as VHDL or Verilog. The compilers are able to generate parallel implementations of loops, containing large number of operations with limited data dependencies. Also, the incorporation of concurrency into a design avoids the manual creation of the RTL implementation. Furthermore, much of the debugging and verification can be performed at a high level rather than at the RTL code level. Consequently, a faster design is possible thanks to the reduction of the debugging and verification time.

The design space exploration (DSE), selecting which code optimizations to apply when implementing an algorithm, is a non-trivial task. Several parameters such as resource consumption and performance must be balanced. The performance of the FPGA is modeled by extending the roofline model proposed by William, Watermans and Patterson [1], with aspects related to resource consumption and code optimzations such as pipelining, loop unrolling and parallelization. The current HLS tools are able to provide good estimations of the performance and the resource consumption of a particular algorithm. Taking into account the resources available on the FPGA, a design may be replicated a number of times. The idea is to maximize the global performance, i.e. the product of an optimized design times the number of identical designs fitting into the FPGA area.

On one hand, the optimizations available on the HLS tools allow to obtain a complete range of performance for one algorithm. On the other hand, knowing the resource consumption of each design and the available resources of the target FPGA, allows to estimate the replication level. Combining the HLS estimation of performance and resource utilization, the performance model allows to optimize the global performance.

This paper is organized as follows. Section 1 presents related work. The performance model for FPGAs is described in Section 2. The elaboration of the proposed model is detailed in Section 3. In Section 4 the performance model is applied to window-based image processing. Conclusions are drawn in Section 5.

1. Related Work

Analytic models have been proposed recently for multicore processors. The roofline model [1] proposed in 2008, is a visual performance model that makes the identification of potential bottlenecks easier and provides a guideline to explore the architecture. It has been proved to be flexible enough to characterize not only multicore architectures but also innovative architectures ([2], [3], and [4]). In the GPU community the model has been well accepted([5], [6] and [7]), due to the similarity of GPU architectures and multicore processors. Nevertheless, as modeling FPGAs demands a considerable number of parameters, the model has been considered for FPGAs just in a few cases [8], [9]. On the other hand, FPGA performance models have been already proposed in the past, [10] and specially [11], but the HLS tools were not mature enough at that time to be included in the model.

2. Performance Modeling

The basis of the roofline model can be adapted and extended for FPGAs. The original model expresses the maximum performance of an application running on an architecture as a function of the Computational Performance (*CP*) of the architecture and its memory Bandwidth (*BW*). The *CP* represents the computational power and is originally measured in GFLOPs. The connection between *CP* and *BW* is the so called Computational Intensity (*CI*), which equals the number of operations executed per byte accessed in the memory. The model assumes either the *CP* or the memory *BW* are the limiting factors. Therefore, the maximal attainable performance of an application running on some platform is given by the following equation:

$$Attainable\ Performance\ =\ min(CP,\ CI \times BW) \tag{1}$$

Figure 1 depicts several performance rooflines, which are defined by the hardware specification or obtained through benchmarking. Furthermore, there are other boundaries, called 'ceilings', that can only be overcome if the application exploits the available resources in an efficient way.

The roofline model is obtained from the main features of a multicore architecture. Because FPGAs are a fully programmable technology whereas the architecture of traditional processors is fixed, the reconfigurable architecture doesn't allow an immediate use of the roofline model. In fact, as the target algorithm defines the architecture, the extended model must be constructed for each algorithm. However, the main principles of the roofline model can be adopted, identifying the performance boundary or what optimizations offer highest performance.

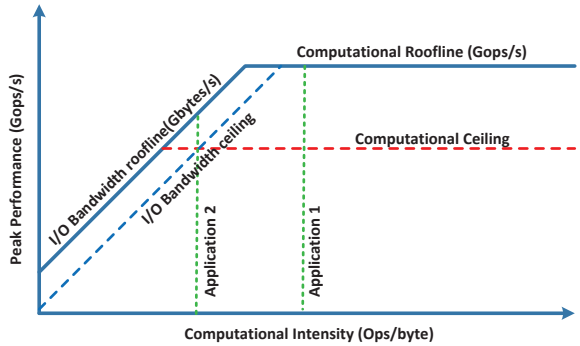

Figure 1. *Basis of the Roofline Model. Example of how two applications with a different CI, are either compute or I/O bound*

2.1. Extending the Roofline Model for FPGAs

The flexibility of the original model allows to adapt the main characteristics for FPGAs. Some, as the operation units or the I/O *BW*, have been already adapted to describe other architectures, but others must be completely adapted or even extended.

Operations: Floating point operations have been widely used in order to show the computational power of microprocessors, but have a prohibitive cost on FPGAs. These operations are often avoided by FPGA designers, adopting other numeric representations as fixed point operations.

I/O Bandwidths: The original roofline model just considers the external memory as memory boundary. Consecutive reviews have increased the range of memory boundaries, by including off-chip memory, L2 Cache, PCIe bandwidth, Network bandwidth,... This wide range of memory *BW*, will be considered as I/O *BW*, can be present at the same time in any architecture, especially in FPGAs.

Roofs and ceilings: Without a fixed architecture, the computational roof is not so evident. A basic operation as one addition can be done using LUTs, FFs or DSPs, masking any performance estimation based on the resource consumption. Additionally, FPGAs may incorporate multiple I/O interfaces, which is defined as multiple I/O ceilings in the model, depending on the algorithm and its implementation.

Scalability: The level of parallelism of an algorithm is defined by the number of operations that can be computed in parallel. On FPGAs, a Processing Element (*PE*) is composed by the operations of the algorithm. The resource consumption of one *PE* defines the maximum number of *PE*s that fits on an FPGA. This value is called scalability (*SC*) and defined as (2). Each replicated *PE* can process input data from independent sources or a set of data from the same source. On the other hand, besides the *SC* is an optimistic value because it does not consider the glue logic between PEs, the impact of these logic resources is introduced into the model thanks to placement and routing.

$$SC = \left\lfloor \frac{Available\ Resources}{Resource\ Consumption\ per\ PE} \right\rfloor \tag{2}$$

Resource Consumption: The flexibility of the FPGAs follows from the fact that they are fully programmable. All the non-dedicated hardware components can be programmed to execute a specific operation. Therefore, the maximum *CP* is determined by the resource consumption of the *PE*s and not by the fixed internal architecture as in other platforms.

2.2. Description of the Extended Model

The original roofline model defines three principal parameters [12] involved on the performance: Computation, Communication and Locality. Three other parameters are directly involved in the extended model: resource consumption, latency and scalability. One by one, the original parameters integrate the new parameters in order to create a more appropriate model for FPGAs.

The *computation*, through the *CI*, reflects the complexity of an algorithm. As was mentioned, the floating point operations are not a good unit for FPGAs, and some kinds of integer operation should be the proper unit. For that reason, byte-operations [Bops], defining the number of operations per byte, is a good candidate. The byte-operations are general enough to cover different kind of integer operations (as fixed-point) and detailed enough to represent the complexity of the algorithms in function of the number of operations per byte. In the extended model, the computational roofline is defined by the properties of the algorithm, such as the level of replication of the algorithm (*SC*) and the attainable throughput of each *PE*. In fact, the maximum number of *PE*s of a particular algorithm, fitting on the available resources of the FPGA, and the attainable performance per PE (CP_{PE}) define the attainable performance. Therefore, the *CP* in (1) is replaced by CP_{FPGA}, where the performance comes from the performance units of the *PE*s and not from the floating-point operations as in the original model.

$$CP_{FPGA} = CP_{PE} \times SC \tag{3}$$

Which, applied to (1) becomes:

$$Attainable\ Performance\ =\ min(CP_{PE} \times SC,\ CI \times BW) \tag{4}$$

Therefore, the model includes the impact of the resource consumption on the performance, through the *SC* and the attainable performance of one *PE*. Furthermore, an increment of the resource consumption leads to a lower operational frequency, decreasing the performance of each *PE*.

The *communication* on the FPGAs can occur in many different ways. The roofline model of an FPGA only includes the available I/O *BW*, which can be represented as different I/O ceilings. Thus, as each communication method has a different *BW*, the model can identify the limiting interface and can show an estimation of the I/O bandwidth using different alternatives. I.e., if the communication with the FPGA is done through the network bus, the attainable performance would increase by using the PCIe bus instead.

The importance of the *locality* resides on the high cost of the communication. In fact, maximizing the locality makes it possible to minimize the communication. In the original roofline model, the locality is possible by the use of cache memory. On FPGAs, however, it can be translated in the use of blocks of memory (BRAMs). BRAMs are an internal dedicated resource of the FPGAs, which offers high *BW*. Thus, by using internal memory for data reuse, the *CI* increases because more operations can be done per external memory access. Also, a higher *CI* means higher available external memory *BW* (by shifting to the right in the roofline model). The loop unrolling in FPGAs is an example of how the internal memory increases the *CI*. The reuse of the input data in some algorithms by unrolling loops, reduces the external memory accesses and increments the *CI*. Finally, the BRAMs must be considered as any other resource on FPGAs, and thus, can be a limiting factor for the final performance as any other logic resource.

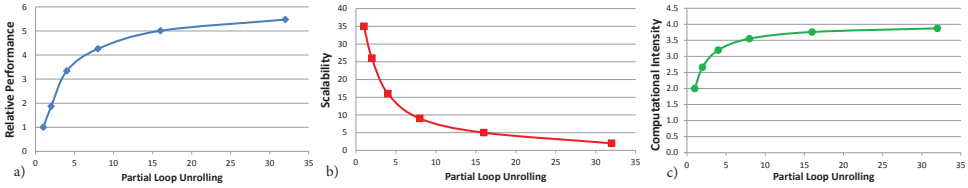

Figure 2. *Impact of loop unrolling over the CI, the SC and the Performance. Generated from the content of Table 1, what is the most beneficial loop unrolling level?*

3. Introducing the High-Level Synthesis tools to construct the model

On FPGAs, only the available logic resources and the I/O *BW* are known in advance. In contrast to the original model, the new model can only be elaborated for an algorithm, and even, for different designs of the same algorithm, because the implementation of the algorithm defines the resource consumption and the attainable performance. In the past, to elaborate an analytical performance model would require a large amount of effort rewriting the HDL algorithm description. However, thanks to the HLS tools, this task can be done much faster and easier nowadays since most of the tools offer different kinds of optimizations. Based on the Table 1, Figure 2 depicts the impact of the partial loop unrolling over several parameters, showing how challenging the selection between all the available optimizations can be. By increasing the *CI* applying loop unrolling, the increment on the resource consumption reduces the number of fittable *PE*s as well. However, there are a direct relation between the resource consumption and the offered performance per *PE*. Therefore, the question is what is the right level of loop unrolling in order to obtain the highest performance. The proposed model offers not only a visual performance estimation but also a guideline to reach the maximum performance through the available optimizations.

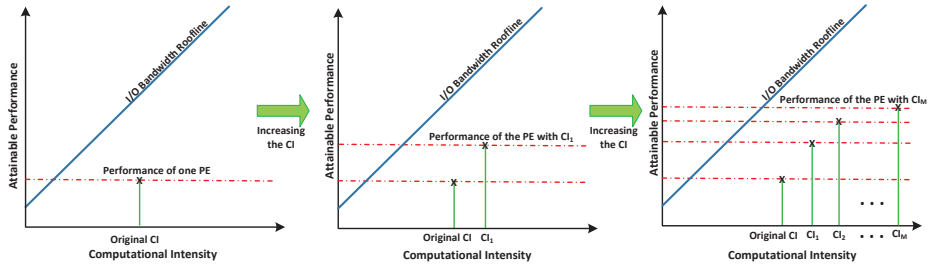

Figure 3. *Impact of the increment of the CI in the extended model. The increment of the CI, thanks to the HLS optimizations for example, increases the performance of one PE as well.*

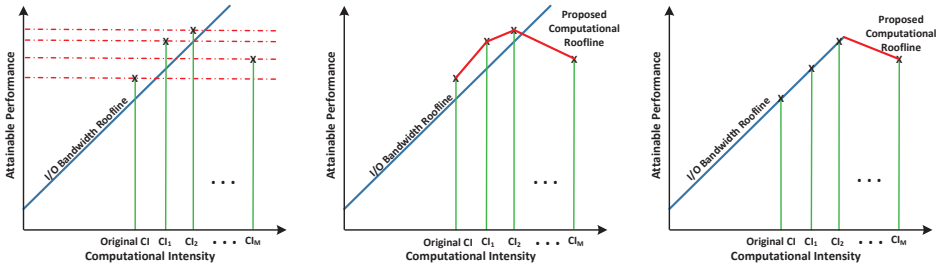

Figure 4. *Impact of the SC on the maximum performance. The attainable performance is limited by the I/O bandwidth as well as the SC of the design. Therefore, the benefit of a high CI is not so evident when the SC is also considered.*

Resource Consumption	No Unrolling	Unrolling x2	Unrolling x4	Unrolling x8	Unrolling x16	Unrolling x32
Slice Registers(301440)	3652	6145	11132	21109	40573	79979
Slice LUTs (150720)	3157	4281	6335	10814	20189	37634
LUT-FF Pairs (37680)	1069	1435	2245	3805	7193	13068
BRAM/FIFO (416)	1	2	3	4	8	15
DSP48 (768)	18	24	36	60	108	204
Max. number of *PE*s	35	26	16	9	5	2
Computational Intensity (CI)	1.9768	2.624	3.144	3.496	3.704	3.816
Throughput per *PE* [GBops/s]	0.636	1.191	2.132	2.711	3.192	3.482
Computational Performance (CP_{FPGA}) [GBops/s]	22.24	30.96	34.08	24.4	15.92	6.96
x8 PCIe BW [GBops/s]	8.302	11.027	13.190	14.678	15.554	16.0272
Resultant Roofline	**8.302**	**11.027**	**13.190**	**14.678**	**15.554**	**6.96**

Table 1. *Generation of the extended model based on the resource consumption, the computational performance of one PE and the I/O limited performance. The resultant roofline is obtained applying the Eqs. (2) and (4) for only one FPGA.*

The introduction of the HLS tools into the model elaboration is done through the DSE of an algorithm. Most of the HLS tools offer valuable information (resource consumption, latency, throughput, ...) required to obtain the extended performance model. Figures 3 and 4 show how to construct the proposed model for an hypothetical algorithm. Figure 3 depicts how the *CI* increases by using HLS optimizations or by using internal memory to reuse data. The *PE*s with higher *CI* offer better performance, but their resource consumption demands increase as well. The inclusion of the resource consumption into the performance model is done in Figure 4, showing how to obtain the computational roofline. One way to obtain the roofline is to consecutively increment the *CI* by applying loop unrolling for example. For each *CI*, the implementation of the target algorithm offers a specific performance and a resource consumption. On one hand, the performance is obtained from the latency of the *PE* and the maximum operational frequency. On the other hand, considering the available resources of the FPGA, the parallelism of the algorithm is exploited by replicating the design as on the GPU approach. This can be done replicating some internal operations of the algorithm, demanding an inherent parallelism, or the replication of the whole algorithm considered as a *PE*. In both cases the model can be applied. Therefore, applying Eq. (3), the *SC* of the design together with the performance of each *PE*, define the value of the maximum attainable performance of the algorithm at each specific *CI* point. In this way, it is possible to obtain the maximum performance for each *CI* point reachable by the algorithm. It is interesting to notice that the attainable performance of one *PE* increases with the *CI* till some point, where the resource consumption starts to be the limiting factor. Therefore, higher *CI* does not need to be necessarily imply higher attainable performance.

4. Experimental Results

The proposed model is applied to an image processing algorithm together with Riverside Optimizing Compiler for Configurable Computing (ROCCC) [13]. The algorithm is implemented in a streaming fashion in a platform composed of two FPGAs Virtex6-LX240 on a Pico Computing backplane board EX500. The backplane has a 16 lane PCIe bus and

accommodates 2 FPGAs, each with an 8 lane PCIe interface. The measured streaming bidirectional *BW* is 4.2GB/s and 5.5 GB/s respectively. Therefore, while the CP_{FPGA} is obtained from the available resources of one FPGA, the available stream PCIe *BW* would be I/O bound.

The implemented algorithm is a morphological operation called dilation. It is a basic example of a computation that uses a moving window over a two-dimensional data structure. Given an input image and a rectangular mask that contains ones and zeros, the output image is determined by placing the mask over the corresponding input pixel and determining the maximum value of the pixels which correspond to the positions of the mask containing ones. Dilation may be categorized as a neighborhood to pixel algorithm. To simplify matters, we consider a square mask of size 3 by 3, containing only ones and the input images are in grayscale. Therefore, it is not necessary to read the mask values. As eight comparisons must be done to generate each output pixel, and nine input pixels need to be read, the initial *CI* of this algorithm equals to $\frac{8}{10}$.

ROCCC offers a type of internal memory, called *smart buffers*, that avoids repeated memory accesses by reusing data [14]. This inherent optimization, which is always active, recognizes memory accesses patterns. For a square mask of 3 by 3 elements, 9 pixels fetch the smart buffers in order to compute the pixel of the first column. For the rest of the columns, only 3 memory accesses are required thanks to the reusing of the prefetched pixels. This reuse of data increases the *CI* of the dilation operation. Therefore, knowing how the smart buffers operate, it is possible to obtain the new *CI*. As the mask is full of ones, the *CI* is defined as follows:

$$CI = \frac{Byte\ Operations}{\left(\frac{Nof\ Memory\ Accesses}{Nof\ Bytes\ of\ the\ Image}\right)} \tag{5}$$

$$CI_{ROCCC} = \frac{8}{\left(\frac{H\times(k^2+1)+H\times(W-1)\times(k+1)}{H\times W}\right)} = \frac{8\times W}{(k^2+1)+(W-1)\cdot(k+1)} \tag{6}$$

Here H and W are the height and the width of the image respectively. Since we are assuming a square mask, k represents both dimensions of the dilation kernel, but the formula can be adjusted for non-squared kernels. The first term of the denominator reflects the pre-fetching of the smart buffers while the other adder shows the additional fetches in the remaining steps.

In addition to the smart buffers, ROCCC offers other optimizations, such as loop unrolling, which is able to increase the *CI* by reducing the memory accesses. Extending the Eq. (6), a generalized version considering the partial loop unrolling impact over the *CI* can be obtained:

$$CI_{PLU} = \frac{8\times N_{PLU}\times W}{((N_{PLU}+k-1)\times k+N_{PLU})+(W-1)\times((N_{PLU}+k-1)+N_{PLU})} \tag{7}$$

Here N_{PLU} represents the level of loop unrolling. Figure 2(c) also shows how the memory accesses are reduced and the *CI* is increased. I.e., by unrolling the loop 2 times the *CI* increases up to 2.56, and if the unroll is unrolled further, the *CI* approaches 4.

Table 1 summarizes the elaboration of the proposed model. The VHDL code generated by ROCCC is synthesized using the Xilinx ISE 14.4 design software to obtain the resource consumption. Once the resource consumption is obtained for each design, the

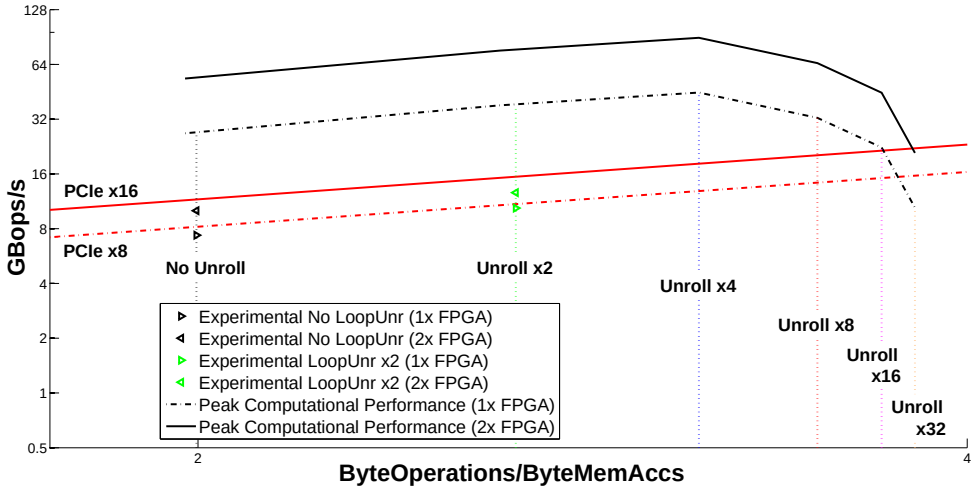

Figure 5. *Superimposed performance models of one FPGA (dashed lines) and two FPGAs (continuous lines) of dilation using ROCCC.*

most limiting resource is identified and the maximum number of *PE*s which can fit on the FPGA is estimated. The *CI* is obtained from the number of memory accesses and knowing that each output pixel requires 8 Bops. Finally, the reachable throughput of each *PE* is obtained from the total latency operating in pipeline. Therefore, with both parameters, it is possible to derive the maximum attainable CP_{FPGA}. The minimum of the CP_{FPGA} and the I/O *BW* limited performance for each *CI* defines the performance model.

Figure 5 shows the roofline obtained from the measurements. Instead of removing all the computational boundaries above the I/O bound, we prefer to keep it in order to clarify the proposed model. As is depicted, the *CI* increases not only due to *smart buffers* but also with the loop unrolling optimization. The vertical lines depicted on 5 represent the obtained *CI* due to both optimizations. This increment is beneficial since more I/O *BW*, which is the limiting factor, can be achieved. However, by further unrolling the loop, the latency of the operations as well as the resource consumption increase due to the internal memory consumption. In fact, after unrolling 16 times the resource consumption of each *PE* is so high that the attainable performance drops below the I/O limited performance.

5. Conclusions

In this paper we have proposed a performance model which extends the roofline model for FPGAs. By analysing the main characteristics of the roofline model we missed the connection between the CP_{FPGA} and the resource consumption, one of the most important parameters on FPGAs. As solution, the proposed model uses the HLS tools to combine the basis of the roofline model with the main characteristics of FPGAs. Finally, we have applied the model to a window-based image processing algorithm using ROCCC. Our next steps are the automatization of the construction of the model and the exploration of the model with more complex algorithms.

References

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures", *Commun. ACM*, vol. 52, no.4, Apr. 2009.

[2] Y. Sato, R. Nagaoka, A. Musa, R. Egawa, H. Takizawa, K. Okabe, and H.Kopbayashi, "Performance tuning and analysis of future vector processors based on the roofline model", *10th workshop on Memory performance: Dealing with Applications, systems and architecture*, ACM, pp. 7-14, 2009

[3] M. Reichenbach, M. Schmidt, and D. Fey, "Analytical model for the optimization of self-organizing image processing systems utilizing cellular automata", *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, IEEE, pp.162-171, 2011

[4] J.A. Lorenzo, J.C. Pichel, T.F. Pena, M. Suarez, and F.F. Rivera, "Study of Performance Issues on a SMP-NUMA System using the Roofline Model", *PDPTA Int. Conf., Las Vegas (USA)*, 2011

[5] H. Jia, Y. Zhang, G. Long, J. XU, S. Yan, and Y. Li, "GPURoofline: a model for guiding performance optimizations on GPUs", *18th international conference on Parallel Processing*, Euro-Par'12, pp. 920-932, 2012

[6] K. Ki-Hwan, K. KyoungHo, and P. Q-Han, "Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model", *Computer Physics Communications, Elsevier*, vol.182, no. 6, pp. 1201-1207, 2011

[7] C. Nugteren and H. Corporaal, "The boat hull model: adapting the roofline model to enable performance prediction for parallel computing", *17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pp. 291-292, 2012

[8] M. Spierings and R.van de Voort, "Embedded platform selection based on the Roofline Model: Applied to video content analysis", 2012

[9] B. da Silva, A. Braeken, E.H. D'Hollander, A. Touhafi, J.G. Cornelis and J. Lemeire, "Performance and toolchain of a combined GPU/FPGA desktop", *FPGA*, pp. 274, 2013

[10] P. Joonseok, P.C. Diniz, K.R.S. Shayee, "Performance and Area Modeling of Complete FPGA Desings in the Precense of Loop Transformations", *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1420-1435, 2004

[11] L. Deng, K. Sobti, Y. Zhang, C. Chakrabarti, "Accurate Area, Time and Power Models for FPGA-Based Implementations", *Journal of Signal Processing Systems*, no. 63, pp. 39-50, 2011

[12] S. Williams, D. Patterson, L. Oliker, J. Shalf, and K. Yelick, "The roofline model", *Performance Tuning of Scientific Applications*, pp. 195-215, CRC, 2010

[13] J. Villarreal, A. Park, W. Najjar and R. Halstead, "Designing modular hardware accelerators in C with ROCCC 2.0", *In Field-Programmable Custom Computing Machines (FCCM)*, 18th IEEE Annual International Symposium on (pp. 127-134), 2010.

[14] Z. Guo, B. Buyukkurt, and W. A. Najjar, "Input data reuse in compiling window operations onto reconfigurable hardware", *LCTES*, pp. 249-256, 2004