RESSALVA

Atendendo solicitação do
autor, o texto completo
desta dissertação será
disponibilizado somente a
partir de 28/02/2020.

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Gustavo Leite

Performance Evaluation of Code Optimizations in FPGA Accelerators

São José do Rio Preto

2019

Gustavo Leite

Performance Evaluation of Code Optimizations in FPGA Accelerators

Orientador: Prof. Dr. Alexandro José Baldassin

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiamento: CAPES e FAPESP (Proc.: 2017/09065-9 e 2018/08116-1)

São José do Rio Preto

2019

L533p

Leite, Gustavo
Performance Evaluation of Code Optimizations in FPGA Accelerators / Gustavo Leite. -- São José do Rio Preto, 2019
66 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Alexandro José Baldassin

1. FPGA. 2. High-Performance Computing. 3. Code Optimization. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Gustavo Leite

Performance Evaluation of Code Optimizations in FPGA Accelerators

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiamento: CAPES e FAPESP (Proc.: 2017/09065-9 e 2018/08116-1)

Comissão Examinadora

- Prof. Dr. Alexandro José Baldassin (Orientador)
  Departamento de Estatística, Matemática Aplicada e Computação
  Universidade Estadual Paulista – UNESP
- Prof. Dr. Orlando de Andrade Figueiredo
  Departamento de Estatística, Matemática Aplicada e Computação
  Universidade Estadual Paulista – UNESP
- Prof. Dr. Emilio de Camargo Francesquini
  Centro de Matemática, Computação e Cognição
  Universidade Federal do ABC – UFABC

Rio Claro

28 de agosto de 2019

To my family and my professors

Acknowledgements

Thanks to Professor Alexandro Baldassin for his guidance, inspiration, immeasurable patience and all the opportunities he created for me.

Thanks to my family Sandra Regina Vieira, Sergio Luiz Leite, Bruna Karina Leite, Alexandre Locatelli Bertoline Filho, Amanda Tofolo, Carla Cristiane Garutti and Daniel Basilio Dias for the unconditional support.

Thanks to all the professors from the Department of Statistics, Applied Mathematics and Computer Science (DEMAC) for helping me grow not only intellectually but also as a person. Thanks to Professor José Nelson Amaral for the contributions and guidance during my visit to the Computing Science Department in the University of Alberta. Thanks to Professor Guido Araújo for the insightful comments and contributions. Thanks to professors Daniel Carlos Guimarães Pedronette, Ivan Rizzo Guilherme and Orlando de Andrade Figueiredo for inspiration.

Thanks to the defense committee for the contributions.

Thanks to all my friends from Laboratório de Inteligência Artificial Aplicada ao Petróleo (LIAAP), University of Alberta Systems Laboratory (UASYS) and also my friends from Araras-SP and Rio Claro-SP for support. Thanks to the Department of Statistics, Applied Mathematics and Computer Science (DEMAC), the São Paulo State University (UNESP) and the University of Alberta.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This study was also funded by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), grants 2017/09065-9 and 2018/08116-1.

Resumo

Com o crescimento contínuo do consumo de energia em microprocessadores, cientistas e engenheiros da computação redirecionaram atenção a arquiteturas heterogêneas, onde dispositivos de classes diferentes são usados para acelerar a computação. Dentre eles, existem as FPGAs (*Field-Programmable Gate Arrays*) cujo hardware pode ser reconfigurado após sua fabricação. Esta classe de dispositivos demonstra desempenho comparável aos processadores convencionais enquanto consomem apenas uma fração de energia. O uso de FPGAs vem se proliferando nos últimos anos e a perspectiva é que o nível de adoção continue a crescer. No entanto, programar FPGAs e aprimorar os programas para obter maior desempenho continua uma tarefa não trivial. Este trabalho apresenta uma compilação das principais transformações de código para otimização de programas direcionados à FPGAs. Neste trabalho também é avaliado o desempenho de programas executando em FPGAs. Mais especificamente, um subconjunto das transformações de código são aplicadas em um kernel OpenCL e os tempos de execução são medidos em um dispositivo da Intel®. Os resultados mostram que, sem a aplicação das transformações, o desempenho dos dispositivos é abaixo do que é observado quando as transformações são de fato aplicadas.

Palavras-Chave: FPGA, computação de alto desempenho, otimização de código.

Abstract

With the ever increasing power wall in microprocessor design, scientists and engineers shifted their attention to heterogeneous architectures, wherein several classes of devices are used for different kinds of computation. Among them are FPGAs whose hardware can be reconfigured after manufacturing. These devices offer comparable performance to CPUs while consuming only a fraction of energy. In fact, the use of FPGAs have been proliferating in recent years and should continue to do so considering the amount of attention these devices are receiving. Still, programmability and performance engineering in FPGAs remain hard. This work presents a compilation of the most prominent code transformations for optimizing code aimed at FPGAs. In this work we also evaluate the performance of programs running on FPGAs. More specifically, we apply a subset of the code transformations to an OpenCL kernel and measure the execution time on a Intel® FPGA. We show that, without applying these transformations before execution, poor performance is observed and the devices are underutilized.

Keywords—FPGA, high-performance computing, code optimization.

List of Figures

Figure 1 – Block diagram of an adaptive logic module from an Intel® Stratix® V device. This particular model exhibits a 8-way adaptive LUT, two full adders and four general-purpose registers (Adapted from: (1)). . . . 21

Figure 2 – Boolean function $Z = A \cdot B + \bar{C}$ implemented using a 3-way lookup table. The inputs to the combinational logic A, B, C are connected to the control lines of a multiplexer which selects the corresponding bit of the "table" stored in the D-latches. 22

Figure 3 – Simplified organization of an FPGA chip: on the edge are the input/output blocks (IO); the logic elements (LE) do the actual computing, as shown in Figure 1; there are also programmable switches (PS) and connect boxes (CB) for routing the I/O blocks to the logic elements. . 23

Figure 4 – Compilation flow of OpenCL code for FPGAs. 25

Figure 5 – OpenCL Platform Model (Adapted from: (2)). 26

Figure 6 – OpenCL NDRange Execution Model (Adapted from: (2)). 27

Figure 7 – OpenCL Memory Model (Adapted from: (2)). 28

Figure 8 – OpenCL Programming Model (Adapted from: (2)). 29

Figure 9 – Pipelining data between kernels (Adapted from: (3)). 32

Figure 10 – Aligned versus non-aligned memory. 32

Figure 11 – Replication and SIMD (Adapted from: (4)). 34

Figure 12 – Arrays without and with aliasing, respectively. Note that, when the arrays alias, data in the intersect region belongs to both arrays at the same time. Therefore, a write operation on A can potentially modify B, creating a memory dependence. If the dependence is not present, the arrays can be modified independently. 38

Figure 13 – Flow of optimization proposed by Zohouri *et al.*. In this scheme, the user manually modify the device code by applying transformations by hand. The rest of the workflow continues normally (Adapted from: (5)). 43
Figure 14 – Comparison of speedup and power efficiency between a CPU (Intel® E5-2670), GPU (Nvidia® K20c) and FPGA (Intel® Stratix V) (Source: (5)). 44
Figure 15 – Flow of transformations proposed by Lloyd *et al.*. In this scheme, information if propagated between host and device compiler. In order to decide if a transformation can be applied in device code, the compiler has to perform an analysis on the host first. (Adapted from: (6)). 45
Figure 16 – Inferred flow of optimization proposed by Lee *et al.*. In this scheme, the user writes OpenACC annotated code and the OpenARC compiler generates optimized host and devices sources automatically. (Adapted from: (3)). 47
Figure 17 – Visual representation of the matrix multiplication operation $C = AB$. Element $c_{i,j}$ equals the vector product between the i-th row in A and the j-th column in B. 51
Figure 18 – Aligned versus unaligned data transfer. 54
Figure 19 – Matrix multiplication kernel with loop unrolling. The graph shows unroll factor versus execution time. Matrices A and B sizes are 1024×1024 elements and the computation was divided into 32 work-items per work group. 55
Figure 20 – Matrix multiplication kernel with compute unit replication. The graph shows number of units versus execution time. Matrices A and B sizes are 1024×1024 elements and the computation was divided into 16 work-items per work group. 57

Figure 21 – Matrix multiplication kernel with compute unit replication and loop unrolling. The graph shows number of units versus execution time. The unroll factor is fixed at 2 for every kernel. Matrices A and B sizes are 1024×1024 elements and the computation was divided into 16 work-items per work group. 59
Figure 22 – Our proposed compilation flow that takes OpenMP annotated code, perform source-to-source transformations and generates the host and devices sources that can finally be compile separately using standard compiler tools. 62

List of Tables

Table 1 – Summary of code transformations studied in previous works. 48
Table 2 – Area usage of matrix multiplication kernels with loop unrolling and parameters marked with `restrict` keyword. No significant difference in area usage was observed when compiling without `restrict`. . . . 56
Table 3 – Area usage of matrix multiplication kernels with compute unit replication and parameters marked with `restrict` keyword. No significant difference in area usage was observed when compiling without `restrict`. 58
Table 4 – Area usage of matrix multiplication kernels with compute unit replication, loop unrolling (factor 2) and parameters marked with `restrict` keyword. No significant difference in area usage observed when compiling without `restrict`. 60

List of abbreviations and acronyms

AI	Artificial Intelligence
ALM	Adaptive Logic Module
ALUT	Arithmetic and Logic Unit
API	Application Programming Interface
ARM	Advanced RISC Machine
CB	Connect Box
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CPU	Central Processing Unit
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
FF	Flip Flip
FPGA	Field-Programmable Gate Array
GCC	GNU Compiler Collection
GPU	Graphic Processing Unit
HARP	Hardware Accelerator Research Program
HDL	Hardware Description Language
HLS	High-Level Synthesis
I/O	Input/Output
IC	Integrated Circuit
IR	Intermediate Representation
LLVM	Low Level Virtual Machine
LUT	Lookup Table
MPI	Message Passing Interface
OpenCL	Open Computing Language
OpenMP	Open Multi Processing
PE	Programmable Element

PLD	Programmable Logic Devices
PL	Programmable Logic
PROM	Programmable Read-Only Memory
PS	Programmable Switch
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
TPC	Thread Pool Composer
TPU	Tensor Processing Unit
VHDL	VHSIC Hardware Description Language

Contents

1 Introduction 16

1.1 Motivation 17

1.2 Objectives 18

1.3 Text Organization 19

2 Background 20

2.1 Field-Programmable Gate Arrays 20

2.2 OpenCL Architecture 26

2.2.1 Platform Model 26

2.2.2 Execution Model 27

2.2.3 Memory Model 28

2.2.4 Programming Model 29

3 Code Optimizations and Related Work 31

3.1 Code Optimizations 31

3.1.1 Kernel Pipelining 31

3.1.2 Direct Memory Access Alignment 32

3.1.3 Loop Unrolling 33

3.1.4 Replication and SIMD 33

3.1.5 Reduction with Shift Registers 35

3.1.6 Sliding Window 36

3.1.7 Single Work-Item Conversion 37

3.1.8 Restrict Parameters 38

3.1.9 Loop Collapse 38

3.1.10 Branch Variant Code Motion 39

3.1.11 Static Memory Coalescing 40
3.1.12 Discussion 41
3.2 Related Research 42

4 Experimental Analysis 50
4.1 Materials 50
4.2 Results 51
4.2.1 Memory transfer 53
4.2.2 Loop Unrolling 53
4.2.3 Compute Unit Replication 56
4.2.4 Loop Unrolling + Compute Unit Replication 58
4.2.5 Discussion 58

5 Conclusion 61
5.1 Future Work 61

REFERENCES 63

1 Introduction

With Moore's Law (7) and Dennard Scaling (8) approaching their end, the high-performance market shifted its focus towards heterogeneous computing systems, where each device is specialized to accelerate domain-specific applications (9). Among the many classes of devices, the most noteworthy are graphical processing units (GPUs), tensor processing units (TPUs), and field-programmable gate arrays (FPGAs) (10). In particular, FPGAs have existed since the mid-1980s and have been used to create logic circuits for embedded systems. More recently, FPGAs also have been commercialized as accelerators, integrating multicore ARM-processors, DRAM, digital signal processors (DSPs) and storage onto a single board usually referred to as "System on Chip" (SoC). Due to their reconfigurable nature, FPGAs offer comparable performance compared to CPUs and usually higher performance per watt compared to GPUs.

Power efficiency is listed in various technical reports as the top hardware challenge in the road to achieve exascale computing (11, 12, 13). According to a report published by U.S. Department of Energy (11), the cost of ownership of a petascale computer sits between $5-10M annually. For exascale computers, however, this cost can reach $2.5B per year with current technology. With that in mind, cloud providers and other players in the HPC market started investigating reconfigurable hardware as a way to make exascale computing feasible. This will be critical going forward considering that electricity and cooling equipments costs sum up to 26% of the total cost of ownership of a data center's infrastructure (14). Amazon Web Services, for instance, one of the leading cloud providers, recently upgraded their servers and now offer instances with up to 8 FPGAs (15, 16). Microsoft Research is using FPGAs for running deep neural networks for real time AI (17).

1.1 Motivation

Despite the clear benefits, FPGA devices are still not as widespread as GPUs, for instance. This is due to three main reasons: (i) the long time to perform hardware synthesis; (ii) the lack of a high-level programming model; and (iii) lack of portability. The process of translating a circuit written in hardware description language into a bitstream—a hardware configuration file—is called hardware synthesis. Synthesizing hardware can last from minutes to a few days (18). This characteristic limits the ability to write incremental code and fast prototyping. Although the programmability of these devices has improved since their first appearance, current software development kits adopted OpenCL (2). For novices and non-experts, even that can be fairly low-level. Before that, reconfigurable hardware was mainly programmed using hardware description languages (VHDL, Verilog), which was even worse since these languages work in the register transfer level (RTL), a paradigm most software developers are unfamiliar with. Lastly, porting a design from one device to another requires a considerable amount of work to be redone. The challenge is aggravated when porting across devices from different vendors.

Performance tuning is not trivial and usually requires several iterations. With other classes of devices, it is straightforward to experiment with different implementations. With FPGAs, however, the long synthesis time severely delays the design cycle. The challenge is scaled when moved to a cluster—now programmers needs to worry about data transfer and workload balancing among the nodes. With this scenario in mind, it is important that we better understand how to tune applications and evaluate trade-offs.

1.2 Objectives

In this work, we provide a thorough analysis of existing code optimizations aimed at OpenCL applications for FPGAs. We not only show how to apply them, but also explain the architectural features of FPGAs that make these optimizations effective in the first place. We provide a discussion related to these optimizations regarding portability to other devices and the difficulty of tuning its parameters. We also present a research on related works regarding code optimization for FPGA and perform an experimental analysis aimed at evaluating the performance of individual code optimizations. For this matter, we select a subset of optimizations, apply them individually to a naive implementation of a matrix multiplication and measure the execution time. In particular, this work makes the following contributions:

1. It compiles and explains a collection of code optimizations found in the literature that can benefit the performance of OpenCL applications running on Intel® FPGAs;
2. It analyzes and summarizes the current state of research about FPGAs for high-performance computing;
3. It presents a performance evaluation of a subset of these transformations on a real application.

We stress the importance of this work in upcoming years because adoption of FPGA devices is rising but not enough programmers are capable of working with such devices. Unlike GPUs which are widespread and have a broad set of open-source tools for profiling and characterization, FPGA development is still in its infancy and the tools are mainly proprietary. There is also the difficulty of running experiments due to the long compilation time, which poses a barrier to thorough characterization of performance. Therefore, this work serves as a first step towards understanding the performance of

code optimizations for FPGAs with the aim of automating them through a compiler infrastructure.

1.3 Text Organization

This document is organized as follows: we present the necessary background about FPGAs and OpenCL in Chapter 2; on Chapter 3 we present code optimizations found in the literature and also the related work that evaluate some of these optimizations; on Chapter 4 we present and discuss experimental data gathered by writing benchmarking applications; finally, on Chapter 5 we present our final considerations and plans for future work.

5 Conclusion

The challenge of increasing performance-per-watt in high-performance computing systems led engineers to adopt reconfigurable accelerators. However, most programmers are unfamiliar with this class of devices, therefore the task of optimizing applications for FPGAs is not trivial. One of the reasons for this difficulty is the lack of performance portability in OpenCL kernels. In other words, a kernel written for GPUs may show low performance when executed on FPGAs. In that sense, this work compiles and explains a collection of code optimizations available in the literature that can benefit code targeted at reconfigurable accelerators. Besides that, this work contribute with a summary of the current stat of research about FPGAs in high-performance computing and also quantifies the performance gain of some of these optimizations. This work provides the first step towards a full performance characterization for FPGAs that will later be used to build compiler-based tools for automatic optimization.

5.1 Future Work

As a next step we envision a directive-based solution for executing high-performance applications on FPGAs. The solution, based on the Clang/LLVM compiler infrastructure, should take an OpenMP code with target directives, optimize the target region for FPGA execution and finally offload this code to a cloud of reconfigurable accelerators.

A general overview of the approach can be visualized in Figure 22. Starting with the original OpenMP source code, the compiler extracts the target region and apply transformations to it, such as described in Section 3.1. These transformations could be applied in the source level using Clang's LibTooling (39) library. The library allows code

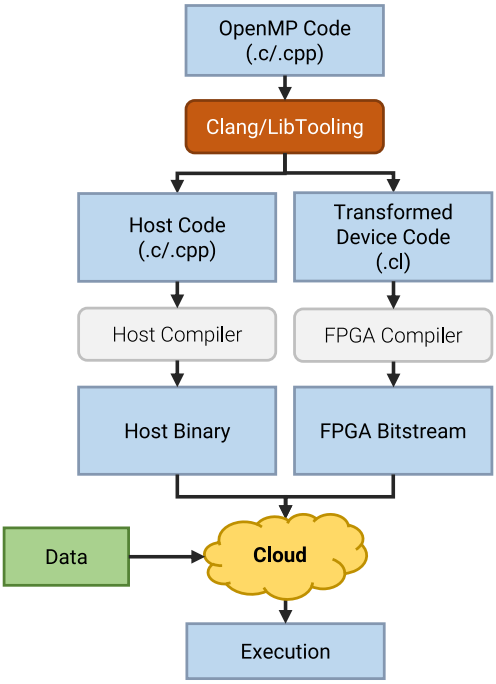

Figure 22 – Our proposed compilation flow that takes OpenMP annotated code, perform source-to-source transformations and generates the host and devices sources that can finally be compile separately using standard compiler tools.

to be inserted, removed or modified at a particular location in the Abstract Syntax Tree (AST). The tree will later be translated back to source code as an OpenCL kernel. On the host side, the compiler will insert runtime calls to the OpenMP target library the same way it is done for other types of accelerators and then continue until an executable binary is produced. Finally, triggering the execution locally will upload the code and data to the cloud, remotely trigger execution and, at the end, transfer the output back to the local machine. This process shall be carried seamlessly, requiring little configuration from the user.

References

1 Intel Corporation. *Intel(R) Arria(R) 10 Device Overview*. [S.l.], 2018.

2 Khronos Group. *Open Computing Language (OpenCL)*. 2019. [Online]. Available: <https://www.khronos.org/opencl/>. (Acessed Feb. 15, 2019).

3 LEE, S.; KIM, J.; VETTER, J. S. OpenACC to FPGA: A framework for directive-based high-performance reconfigurable computing. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. [S.l.: s.n.], 2016. p. 544–554. ISSN 1530-2075.

4 Intel Corporation. *Intel(R) FPGA SDK for OpenCL(TM) Pro Edition: Best Practices Guide*. [S.l.], 2018.

5 ZOHOURI, H. R.; MARUYAMA, N.; SMITH, A.; MATSUDA, M.; MATSUOKA, S. Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs. In: *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. [S.l.: s.n.], 2016. p. 409–420. ISSN 2167-4337.

6 LLOYD, T.; CHIKIN, A.; OCHOA, E.; ALI, K.; AMARAL, J. N. A case for better integration of host and target compilation when using OpenCL for FPGAs. In: *FSP 2017; Fourth International Workshop on FPGAs for Software Programmers*. [S.l.: s.n.], 2017. p. 1–9.

7 MOORE, G. E. Cramming more components onto integrated circuits. *Electronics*, v. 38, n. 8, p. 56–59, abr. 1965.

8 DENNARD, R. H.; GAENSSLEN, F. H.; RIDEOUT, V. L.; BASSOUS, E.; LEBLANC, A. R. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, v. 9, n. 5, p. 256–268, Oct 1974. ISSN 0018-9200.

9 HENNESSY, J.; PATTERSON, D. *Computer architecture: A Quantitative Approach*. Cambridge, MA: Morgan Kaufmann Publishers, 2019. ISBN 0128119055.

10 HENNESSY, J. L.; PATTERSON, D. A. A new golden age for computer architecture. *Commun. ACM*, ACM, New York, NY, USA, v. 62, n. 2, p. 48–60, jan. 2019. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/3282307>.

11 U.S. Department of Energy. *The Opportunities and Challenges of Exascale Computing*. [S.l.], 2010.

12 KOGGE, P.; BERGMAN, K.; BORKAR, S.; CAMPBELL, D.; CARLSON, W.; DALLY, W.; DENNEAU, M.; FRANZON, P.; HARROD, W.; HILLER, J.; KARP, S.; KECKLER, S.; KLEIN, D.; LUCAS, R.; RICHARDS, M.; SCARPELLI, A.; SCOTT, S.; SNAVELY, A.; STERLING, T.; WILLIAMS, R. S.; YELICK, K.; BERGMAN, K.; BORKAR, S.; CAMPBELL, D.; CARLSON, W.; DALLY, W.; DENNEAU, M.; FRANZON, P.; HARROD, W.; HILLER, J.; KECKLER, S.; KLEIN, D.; KOGGE, P.; WILLIAMS, R. S.; YELICK, K. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. [S.l.], 2008.

13 DONGARRA, J.; BECKMAN, P.; MOORE, T.; AERTS, P.; ALOISIO, G.; ANDRE, J.-C.; BARKAI, D.; BERTHOU, J.-Y.; BOKU, T.; BRAUNSCHWEIG, B.; CAPPELLO, F.; CHAPMAN, B.; CHI, X.; CHOUDHARY, A.; DOSANJH, S.; DUNNING, T.; FIORE, S.; GEIST, A.; GROPP, B.; HARRISON, R.; HERELD, M.; HEROUX, M.; HOISIE, A.; HOTTA, K.; JIN, Z.; ISHIKAWA, Y.; JOHNSON, F.; KALE, S.; KENWAY, R.; KEYES, D.; KRAMER, B.; LABARTA, J.; LICHNEWSKY, A.; LIPPERT, T.; LUCAS, B.; MACCABE, B.; MATSUOKA, S.; MESSINA, P.; MICHIELSE, P.; MOHR, B.; MUELLER, M. S.; NAGEL, W. E.; NAKASHIMA, H.; PAPKA, M. E.; REED, D.; SATO, M.; SEIDEL, E.; SHALF, J.; SKINNER, D.; SNIR, M.; STERLING, T.; STEVENS, R.; STREITZ, F.; SUGAR, B.; SUMIMOTO, S.; TANG, W.; TAYLOR, J.; THAKUR, R.; TREFETHEN, A.; VALERO, M.; STEEN, A. V. D.; VETTER, J.; WILLIAMS, P.; WISNIEWSKI, R.; YELICK, K. The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.*, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 25, n. 1, p. 3–60, fev. 2011. ISSN 1094-3420. Disponível em: <http://dx.doi.org/10.1177/1094342010391989>.

14 RASMUSSEN, N. *Determining Total Cost of Ownership for Data Center and Network Room Infrastructure*. [S.l.], 2011.

15 Amazon Web Services. *Amazon EC2 F1 Instances*. 2019. [Online]. Available: <https://aws.amazon.com/ec2/instance-types/f1/>. (Acessed Feb. 11, 2019).

16 BARR, J. *EC2 F1 Instances with FPGAs – Now Generally Available*. 2017. [Online]. Available: <https://aws.amazon.com/blogs/aws/ec2-f1-instances-with-fpgas-now-generally-available/>. (Acessed Feb. 11, 2019).

17 FOWERS, J.; OVTCHAROV, K.; PAPAMICHAEL, M.; MASSENGILL, T.; LIU, M.; LO, D.; ALKALAY, S.; HASELMAN, M.; ADAMS, L.; GHANDI, M.; HEIL, S.; PATEL, P.; SAPEK, A.; WEISZ, G.; WOODS, L.; LANKA, S.; REINHARDT, S. K.; CAULFIELD, A. M.; CHUNG, E. S.; BURGER, D. A configurable cloud-scale dnn processor for real-time ai. In: *Proceedings of the 45th Annual International Symposium on Computer Architecture*. Piscataway, NJ, USA: IEEE Press, 2018. (ISCA '18), p. 1–14. ISBN 978-1-5386-5984-7. Disponível em: <https://doi.org/10.1109/ISCA.2018.00012>.

18 BACON, D.; RABBAH, R.; SHUKLA, S. FPGA programming for the masses. *Queue*, ACM, New York, NY, USA, v. 11, n. 2, p. 40:40–40:52, fev. 2013. ISSN 1542-7730. Disponível em: <http://doi.acm.org/10.1145/2436696.2443836>.

19 Intel Corporation. *Intel Acquisition of Altera*. 2015. [Online]. Available: <https://newsroom.intel.com/press-kits/intel-acquisition-of-altera/>. (Acessed Feb. 10, 2019).

20 TRIMBERGER, S. M. S. Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *IEEE Solid-State Circuits Magazine*, v. 10, n. 2, p. 16–29, Spring 2018. ISSN 1943-0582.

21 VENUGOPAL, N.; MANIMEGALAI, R. Survey on fpga routing techniques. *International Journal on Computer Science and Engineering*, Engg Journals Publications, v. 4, n. 7, p. 1304, 2012.

22 PRADO, D. F. G. *Tutorial on FPGA Routing*. [S.l.], 2006.

23 GUDISE, V. G.; VENAYAGAMOORTHY, G. K. Fpga placement and routing using particle swarm optimization. In: *IEEE Computer Society Annual Symposium on VLSI*. [S.l.: s.n.], 2004. p. 307–308.

24 KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. ISSN 0036-8075. Disponível em: <http://science.sciencemag.org/content/220/4598/671>.

25 KARYPIS, G.; AGGARWAL, R.; KUMAR, V.; SHEKHAR, S. Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 7, n. 1, p. 69–79, March 1999. ISSN 1063-8210.

26 LU, J.; CHEN, P.; CHANG, C.-C.; SHA, L.; HUANG, D. J.-H.; TENG, C.-C.; CHENG, C.-K. ePlace: Electrostatics based placement using nesterov's method. In: *Proceedings of the 51st Annual Design Automation Conference*. New York, NY, USA: ACM, 2014. (DAC '14), p. 121:1–121:6. ISBN 978-1-4503-2730-5. Disponível em: <http://doi.acm.org/10.1145/2593069.2593133>.

27 MARTIN, G.; SMITH, G. High-level synthesis: Past, present, and future. *IEEE Design Test of Computers*, v. 26, n. 4, p. 18–25, July 2009. ISSN 0740-7475.

28 NANE, R.; SIMA, V.; PILATO, C.; CHOI, J.; FORT, B.; CANIS, A.; CHEN, Y. T.; HSIAO, H.; BROWN, S.; FERRANDI, F.; ANDERSON, J.; BERTELS, K. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 35, n. 10, p. 1591–1604, Oct 2016. ISSN 0278-0070.

29 Intel Corporation. *Intel(R) FPGA SDK for OpenCL(TM) Pro Edition: Programming Guide*. [S.l.], 2018.

30 AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. *Compilers: Principles, Techniques, and Tools*. 2. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321486811.

31 Xilinx Corporation. *SDAccel Programmers Guide*. [S.l.], 2019.

32 Xilinx Corporation. *SDAccel Environment Profiling and Optimization Guide*. [S.l.], 2019.

33 LAMBERT, J.; LEE, S.; KIM, J.; VETTER, J. S.; MALONY, A. D. Directive-based, high-level programming and optimizations for high-performance computing with FPGAs. In: *Proceedings of the 2018 International Conference on Supercomputing*. New York, NY, USA: ACM, 2018. (ICS '18), p. 160–171. ISBN 978-1-4503-5783-8. Disponível em: <http://doi.acm.org/10.1145/3205289.3205324>.

34 CHE, S.; BOYER, M.; MENG, J.; TARJAN, D.; SHEAFFER, J. W.; LEE, S.; SKADRON, K. Rodinia: A benchmark suite for heterogeneous computing. In: *2009 IEEE International Symposium on Workload Characterization (IISWC)*. [S.l.: s.n.], 2009. p. 44–54.

35 FEIST, T. *Vivado Design Suite*. [S.l.], 2012.

36 OpenACC. *OpenACC: Directives for Accelerators*. 2019. [Online]. Available: <https://www.openacc.org/>. (Acessed Feb. 15, 2019).

37 LEE, S.; VETTER, J. S. Openarc: Open accelerator research compiler for directive-based, efficient heterogeneous computing. In: *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*. New York, NY, USA: ACM, 2014. (HPDC '14), p. 115–120. ISBN 978-1-4503-2749-7. Disponível em: <http://doi.acm.org/10.1145/2600212.2600704>.

38 Intel Corporation. *Intel(R) Stratix(R) V Device Overview*. [S.l.], 2015.

39 Clang. *LibTooling – Clang Documentation*. 2019. [Online] Available: <https://clang.llvm.org/docs/LibTooling.html>. (Accessed Feb. 17, 2019).