Evaluation of Stencil Based Algorithm Parallelization over System-on-Chip FPGA Using a High Level Synthesis Tool

Luis Castano-Londono[1,3(✉)], Cristian Alzate Anzola[1], David Marquez-Viloria[1], Guillermo Gallo[2], and Gustavo Osorio[3]

[1] Department of Electronics and Telecommunication Engineering, Instituto Tecnológico Metropolitano ITM, Medellín, Colombia
{luiscastano,davidmarquez}@itm.edu.co, cristianalzate224500@correo.itm.edu.co
[2] Rynova Research Group, Rymel Company, Medellín, Colombia
guillermogallo@rymel.com.co
[3] Department of Electrical, Electronics and Computing Engineering, Universidad Nacional de Colombia, Manizales, Colombia
gaosorio@unal.edu.co

Abstract. Iterative stencil computations are present in many scientific and engineering applications. The acceleration of stencil codes using parallel architectures has been widely studied. The parallelization of the stencil computation on FPGA based heterogeneous architectures has been reported with the use of traditional RTL logic design or the use of directives in C/C++ codes on high level synthesis tools. In both cases, it has been shown that FPGAs provide better performance per watt compared to CPU or GPU-based systems. High level synthesis tools are limited to the use of parallelization directives without evaluating other possibilities of their application based on the adaptation of the algorithm. In this document, it is proposed a division of the inner loop of the stencil-based code in such a way that total latency is reduced using memory partition and pipeline directives. As a case study is used the two-dimensional Laplace equation implemented on a ZedBoard and an Ultra96 board using Vivado HLS. The performance is evaluated according to the amount of inner loop divisions and the on-chip memory partitions, in terms of the latency, power consumption, use of FPGA resources, and speed-up.

Keywords: Stencil computation · Field programmable gate array (FPGA) · System-on-a-chip (SoC) · High-Level Synthesis (HLS)

1 Introduction

Iterative stencil computations are present in scientific and engineering applications such as: numerical integration of partial differential equations [11,18,26],

J. C. Figueroa-García et al. (Eds.): WEA 2019, CCIS 1052, pp. 52–63, 2019.
https://doi.org/10.1007/978-3-030-31019-6_5

image processing [3,22], Particle-in-Cell simulation [22], graph processing [15], among others. The acceleration of stencil codes using parallel architectures has been widely studied [2,6,13,17,18,25–28]. One of the most important limitations of the stencil computation is its small operational intensity [23,26], which makes it difficult to take advantage of the supercomputers that have a large number of processing units (microprocessors and GPUs) [23].

In the last years, FPGA based accelerators has been proposed to improve stencil computation performance with low power consumption [5–7,10,16,21,23, 26]. FPGAs have a large number of registers, which facilitates the transfer of data between the iterations of a computation without the need to access an external memory. This leads to an increase in operational intensity and processing speed [26]. In a previous work, we presented an evaluation of architectures for stencil computation, in which it is shown that it is possible to reach execution times similar to those of a CPU used as a reference using registers and on-chip memory [1]. However, these architectures were experimentally tested for small mesh sizes due to the resource limitations of the FPGA used.

FPGA accelerators are usually implemented by means of a hardware design language (HDL) [9,10,19,26]. However, HDL designs require extensive knowledge of the hardware [26]. In order to raise the level of abstraction of designs and facilitate implementation, some High-Level Synthesis tools (HLS) has been used as in [8,14,19,20,24,28]. The HLS tools allow to ignore some hardware details, but often deliver solutions less efficient compared to those obtained using HDL [19]. In these cases it is necessary to manually rewrite the code to optimize, for example, memory access [8].

There have been attempts to improve the performance of HLS solutions. For example, in [6], a set of design options have been explored to accommodate a large set of constraints. Most literature works achieve high performance by evading spatial blocking and restricting the input size. On the other hand, in [28], spatial and temporal blocking are combined in order to avoid input size restrictions. It is well known that one of the bottlenecks in the HLS solutions is access to data [4,8]. In this way, it is necessary to optimize memory management. In [8], graph theory is used in order to optimize the memory banking. In [4], a non-uniform partition of the memory is proposed in such a way that the number of memory banks is minimized. Loop pipelining is another key method for optimization in HLS [13]. However, the performance level of the solutions may not be optimal when complex memory dependencies appear. In [12–14], loop pipelining capabilities are improved in order to handle uncertain memory dependencies.

In this document, it is presented a strategy that attacks the HLS optimization problem on two fronts: memory management and loop pipelining. To achieve the task, it is proposed a method to split the mesh in such a way that total latency is reduced using on-chip memory partitioning and pipeline directives. As a case study is used the two-dimensional Laplace equation implemented for two different development systems, the ZedBoard using Vivado Design Suite and the Ultra96 board using Vivado SDx. The performance is evaluated and compared

according to the amount of inner loop divisions and the memory partitions in terms of the latency, power consumption, use of FPGA resources, and speed-up. The rest of the document is organized as follows. In Sect. 2, it is presented the two-dimensional Laplace equation and the approach to its numerical solution by means of finite difference method. In Sect. 3, the details of the implemented stencil computing system are presented. Results are presented in Sect. 4. Finally, the conclusions are given in Sect. 5.

2 Case Study: Two-Dimensional Laplace Equation

Suppose Ω as a domain of R^2 with boundary defined as $\partial\Omega$. The partial differential equation shown in (1) is considered elliptical for all points $(x, y) \in \Omega$.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{1}$$

This expression is known as two-dimensional Laplace equation and it is used to describe the stationary temperature distribution in a two-dimensional region given some boundary conditions. An approach to the numerical solution of this equation is obtained using the finite difference method. The Ω region is discretized in the two dimensions x and y by defining a number of points I and J respectively. This approach is obtained for the iteration $n + 1$ as in (2), considering a uniform distribution of the points in the domain of the solution.

$$u_{ij}^{n+1} = \frac{1}{4}(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}) \tag{2}$$

The implementation of this approach is known as the Jacobi algorithm. For Dirichlet boundary conditions, it is described for a number of iterations N as shown in Algorithm 1.

Algorithm 1. Stencil for the Laplace equation using the Jacobi algorithm.

Input: initial and boundary conditions, mesh size, number of iterations
Output: temperature $u(x, y)$ at iteration N

```
Loop 1: for n ← 0 to N − 1 do
    Loop 1.1: for j ← 1 to J − 2 do
        Loop 1.1.1: for i ← 1 to I − 2 do
            u_{i,j}^{n+1} ← 0.25(u_{i+1,j}^{n} + u_{i−1,j}^{n} + u_{i,j+1}^{n} + u_{i,j−1}^{n})
        end
    end
end
```

3 System Implementation

The implementation was performed for two different development systems, the ZedBoard using Vivado Design Suite and the Ultra96 board using Vivado SDx. In both cases, the Zynq processing system (PS) interacts through of AXI interface and DMA with a custom IP created in Vivado HLS using C language for the programmable logic (PL) section. The ARM core is the host processor where the main application runs over a PetaLinux terminal, and the custom IP core executes the algorithm based on the stencil scheme. This application includes the generation of initial values and the boundary conditions which are stored in BRAM. Then, the number of iterations is defined and the stencil computation function is called. When the stencil algorithm execution is finished, the results become available in the DDR3 RAM and these can be read and saved in a text file with 15 significant digits in decimal format. The block diagram of the system is shown in Fig. 1.

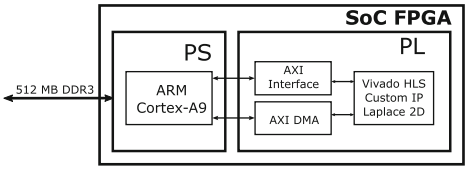

Fig. 1. Block diagram of the system implemented using *Vivado Design Suite*.

3.1 Baseline Architecture of the Custom IP Core

The sequential implementation of the code, defined as architecture A_1, was used as reference to compare against the performances of the parallel implementations. Given that the data arrive to the main function as a vector, the line code for stencil operation in Algorithm 1 is implemented as shown in Algorithm 2.

The maximum size that can be used for a square mesh is determined by the amount of BRAM memory blocks in the FPGA device ($BRAM\_Blocks$). Considering a BRAM block size of 18 Kb, the use of simple floating point format, and that the algorithm requires two arrays to store the values of the last two iterations, the mesh size is calculated as shown in (3).

$$mesh\_size_{max} = \sqrt{\frac{RAM\_Blocks \times 18000}{32 \times 2}} \tag{3}$$

3.2 Parallelization

The acceleration of the algorithm execution, defined as architecture A_2, was achieved using a pipeline directive in Loop 1 of the stencil code. In addition, some modifications are made to the stencil code implementation to reduce the latency.

Algorithm 2. Stencil computation algorithm for N iterations using vectors.

```
   Input: initial values and boundary conditions, mesh size, number of iterations (N)
   Output: temperature at iteration N
 1 Function stencil_2D(u[65536], v[65536]: float) is
 2 |   Loop 1: for j ← 1 to J − 2 do
 3 |   |   Loop 1.1: for i ← 1 to I − 2 do
 4 |   |   |   v[j ∗ X_I + i] ←
   |   |   |   0.25 (u[j ∗ X_I + i + 1] + u[j ∗ X_I + i − 1] + u[(j + 1) ∗ X_I + i] + u[(j − 1) ∗ X_I + i])
 5 |   |   end
 6 |   end
 7 end
 8 Function Laplace_2D() : void is
 9 |   Loop 1: for n ← 0 to N − 1 do
10 |   |   stencil_2D(u,v)
11 |   |   Loop 1.1: for j ← 1 to J − 2 do
12 |   |   |   Loop 1.1.1: for i ← 1 to I − 2 do
13 |   |   |   |   u[j ∗ X_I + i] ← v[j ∗ X_I + i]
14 |   |   |   end
15 |   |   end
16 |   end
17 end
```

A first approach makes the most of the Loop 1.1 of the Laplace function, considering the transfer operations used when the vector u is updated with the vector v to calculate a new iteration. Thus, the upper limit of the external loop is reduced by half as shown in Algorithm 3.

Algorithm 3. Pseudocode of the stencil computation algorithm implementation reducing the number of iterations of the external loop to N/2.

```
   Input: initial and boundary conditions, mesh size, number of iterations (N)
   Output: temperature u(x, y) at iteration N
 1 Function Laplace_2D() : void is
 2 |   Loop 1: for n ← 0 to N/2 do
 3 |   |   stencil_2D(u,v)
 4 |   |   stencil_2D(v,u)
 5 |   end
 6 end
```

To improve performance, a method for splitting the mesh into three blocks on the y axis is proposed, as shown in Fig. 2. The distribution is made so that the number of divisions in block B2 is a power of 2, and considering that the number of rows in blocks B1 and B3 is odd because of the rows of boundary conditions. This distribution allows the application of the parallelization directives in such a way that the synthesis time, the amount of resources and the latency are reduced. The memory partition directive allows the different blocks to access the corresponding data concurrently, which are distributed in a number of smaller arrays defined by the partition factor. The approach, defined as architecture A_3, is described as shown in Algorithm 4.

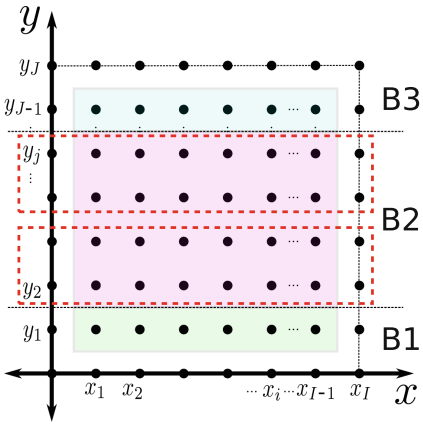

Fig. 2. Distribution of blocks for processing.

Algorithm 4. Pseudocode of the algorithm based on stencil with two-dimensional arrangement of 256×256 for N iterations.

Input: initial and boundary conditions, mesh size ($I \times J$), iterations (N), divisions of $B2$ (PY), rows in B1 (PYF), rows in each subdivision of B2 (PYI), rows in B1 and B2 (PYL)
Output: temperature $u(x, y)$ at iteration N

```
1  Function stencil_2D(u[65536], v[65536]: float) is
2      for j ← 1 to PYF-1 do
3          # pragma pipeline
4          for i ← 1 to I − 2 do
5              s1 ← u[j ∗ X_I + i + 1] + u[j ∗ X_I + i − 1]
6              s2 ← u[(j + 1) ∗ X_I + i] + u[(j − 1) ∗ X_I + i]
7              v[j ∗ X_I + i] ← 0.25 (s1 + s2) ;                    // Stencil for block B1
8          end
9      end
10     for j ← 1 to PY do
11         for i ← 1 to I − 2 do
12             # pragma pipeline
13             for k ← 1 to PYI − 2 do
14                 s1 ← u[(j + PYF + PY ∗ k) ∗ X_I + i + 1]
15                 s2 ← u[(j + PYF + PY ∗ k) ∗ X_I + i − 1]
16                 s3 ← u[(j + PYF + PY ∗ k + 1) ∗ X_I + i]
17                 s4 ← u[(j + PYF + PY ∗ k − 1) ∗ X_I + i]
18                 v[(j + PYF + PY ∗ k) ∗ X_I + i] ← 0.25 (s1 + s2 + s3 + s4) ;   // Stencil
                   for block B2
19             end
20         end
21     end
22     for j ← 1 to PYF do
23         # pragma pipeline
24         for i ← 1 to I − 2 do
25             s1 ← u[(j + PYL) ∗ X_I + i + 1] + u[(j + PYL) ∗ X_I + i − 1]
26             s2 ← u[(j + PYL + 1) ∗ X_I + i] + u[(j + PYL − 1) ∗ X_I + i]
27             v[(j + PYL) ∗ X_I + i] ← 0.25 (s1 + s2) ;             // Stencil for block B3
28         end
29     end
30 end
```

4 Results

The performance of the implemented system was evaluated according to numerical results, execution times, and physical resources of FPGA. Numerical results were obtained for different mesh sizes, from boundary conditions and initial values defined as shown in (4).

$$\begin{cases} u_{xx} + u_{yy} = 0 \\ \quad u = 0, \qquad \forall (x, y) \in \Omega \\ \quad u = 1, \qquad \forall (x, y) \in \partial\Omega \end{cases} \tag{4}$$

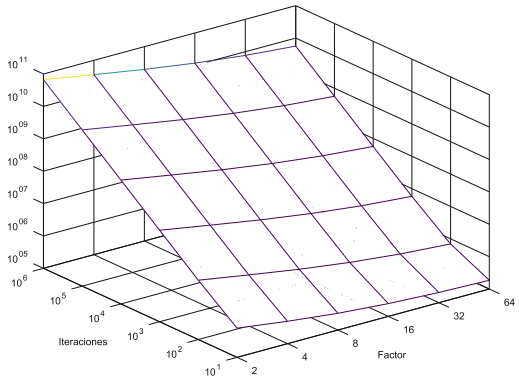

Fig. 3. Latency for 4 processing blocks of the middle division according to number of iterations and partition factor of on-chip memory.

Performances of the implemented architectures are obtained measuring execution time. The architecture A_3 has several configurations, therefore, a design space exploration is performed based on two parameters: number of subdivision in the middle block and the memory partition factor. For this purpose, latencies are obtained in terms of clock cycles for different combinations of both parameters and the number of iterations. The latency measurements are performed for block sizes of 4, 8, 16, 32, and 64 for the middle block, and assigning values of 2, 4, 8, 16, 32, and 64 as memory partition factor. For each combination of these parameters the simulation is carried out for 10^1, 10^2, 10^3, 10^4, 10^5, y 10^6 iterations. In Fig. 3 are shown the latencies for 4 subdivisions of $B2$ based on number of iterations and memory partition factor. It is observed that the performance improves with the increase of this last parameter. Latencies obtained are used for the execution time calculations considering a 100 MHz clock frequency.

Table 1 show the speed-up achieved using the A_3 with 4 processing blocks in relation to the base architecture A_1 and to the sequential execution on CPU. It is observed a number of iterations from which the acceleration tends to a constant value.

Table 1. Speed-up with regards to the base architecture A1 based on number of iterations and memory partition factor.

		Factor											
		t_{A_1}/t_{A_3}						t_{CPU}/t_{A_3}					
		2	4	8	16	32	64	2	4	8	16	32	64
N	10^1	20,44	33,11	49,48	65,72	78,61	87,19	0,87	1,41	2,10	2,79	3,34	3,77
	10^2	23,78	43,15	76,27	123,73	179,63	232,24	0,84	1,53	2,70	4,37	6,35	8,21
	10^3	24,18	44,51	80,67	135,82	206,36	279,03	0,79	1,46	2,64	4,45	6,76	9,14
	10^4	24,22	44,65	81,14	137,16	209,48	284,77	0,79	1,45	2,63	4,45	6,79	9,24
	10^5	24,23	44,66	81,19	137,30	209,80	285,36	0,78	1,44	2,61	4,41	6,74	9,17
	10^6	24,23	44,66	81,19	137,31	209,83	285,42	0,78	1,44	2,62	4,42	6,76	9,19

The best performance was determined making a plot of the latency based on the number of subdivisions in the block $B2$ and memory partition factor for a number of 10^6 iterations, as shown in Fig. 4. The lowest latency was observed using a combination of 4 subdivisions and memory partition factor of 64.

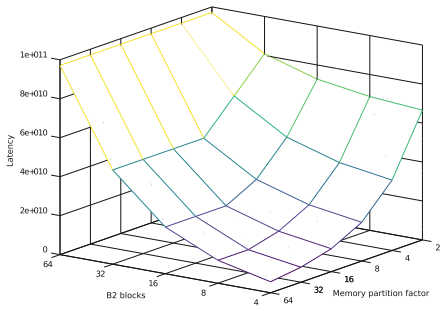

Fig. 4. Latency for 10^6 iterations based on the number of processing blocks and the partition factor of on-chip memory.

Execution times were measured experimentally for the architectures implemented on the ZedBoard and an Ultra96 board. Table 2 shows execution times according to the number of iterations for the implemented architectures.

Table 2. Execution times in microseconds for different number of iterations with the different architectures implemented.

Iterations	ZedBoard				Ultra96	
	A_1	A_2	$A_{3(16\times16)}$	$A_{3(4\times32)}$	$A_{3(4\times64)}$ 100 MHz	$A_{3(4\times64)}$ 200 MHz
10^1	158.212	16.199	3.710	2.086	1.928	993
10^2	1.552.677	150.163	25.259	9.008	7.011	3.561
10^3	15.497.303	1.489.794	240.753	78.231	57.877	29.250
10^4	154.943.576	14.886.115	2.395.667	770.466	566.516	286.139
10^5	1.549.406.305	148.849.317	23.944.818	7.692.819	5.652.881	2.854.981
10^6	15.494.033.580	1.488.481.343	239.436.321	76.916.319	56.516.436	28.543.256

The speedup achieved for the implemented architectures calculated in relation to the baseline and the sequential implementation on CPU is shown in Table 3.

Table 3. Speed-up achieved for the implemented architectures in relation to the sequential implementation on CPU.

Iterations	ZedBoard					Ultra96	
	t_{A_4}/t_{A_1}	A_1	A_2	$A_{3(16\times16)}$	$A_{3(4\times32)}$	$A_{3(4\times64)}$ 100MHz	$A_{3(4\times64)}$ 200 MHz
10^1	75,85	0,04	0,43	1,86	3,31	3,59	6,96
10^2	172,37	0,04	0,38	2,26	6,34	8,15	16,04
10^3	198,10	0,03	0,35	2,20	6,76	9,14	18,08
10^4	201,10	0,03	0,35	2,18	6,79	9,24	18,29
10^5	201,41	0,03	0,35	2,17	6,74	9,17	18,16
10^6	201,44	0,03	0,35	2,17	6,76	9,20	18,21

The consumption of hardware resources for each architecture is shown in Table 4.

Table 4. Hardware resources required on ZedBoard and Ultra96. The entire system includes processing modules.

Resource	ZedBoard				Ultra96	
	A_1	A_2	$A_{3(16\times16)}$	$A_{3(4\times32)}$	$A_{3(4\times64)}$ 100 MHz	$A_{3(4\times64)}$ 200 MHz
LUT	5.644	25.574	16.776	34.277	64.540	64.540
Flip Flop	6.599	22.836	17.689	39.204	69.256	86.570
Slices	2.267	9.100	6407	12.415	–	–
DSP48	5	17	36	97	144	144

The power consumption for the implemented architectures is shown in Table 5.

Table 5. Power consumption for the implemented architectures. The entire system includes processing modules.

	ZedBoard				Ultra96	
	A_1	A_2	$A_{3(16\times16)}$	$A_{3(4\times32)}$	$A_{3(4\times64)}$ 100 MHz	$A_{3(4\times64)}$ 200 MHz
Core Power (W)	0,297	1,005	1,037	1,535	1,349	4,989
Total Power (W)	1,87	2,592	2,617	3,599	3,539	5,341

5 Conclusions

This paper presents a strategy for the implementation of algorithms based stencil on SoC-FPGA using Vivado HLS, addressing the problem of optimization in terms of memory management and parallelization of cycles. The general scheme of the implemented architectures involves the use of an ARM Cortex-A9 microprocessor that acts as master, on which the main application is executed. The processor interacts through an AXI interface with an IP created in Vivado HLS, which performs the execution of the algorithm based on stencil. The architectures are implemented on a ZedBoard Zynq Evaluation and Development Kit under the Vivado Design Suite environment and on an Ultra96 board using Vivado SDx. The source code of the main application is made in C and executed under PetaLinux on the PS using a terminal console. The communication is done using an AXI interface and direct access to memory (DMA).

To improve performance in terms of execution time a method is proposed to split the mesh into three parts on the y-axis. The distribution is done so that the number of rows in block B2 is a multiple of a power of 2, considering that blocks B1 and B3 have one row less because they include contour conditions. An unrolling of the internal cycle is proposed so that the latency of the intermediate cycle is reduced according to the number of subdivisions of B2. Additionally, the on-chip memory partition is made in such a way that each subdivision can access the corresponding data concurrently.

An exploration of the design space for the generalized architecture is performed, based on the number of B2 processing subdivisions and the factor used for the memory partition. For this, latencies are obtained in terms of clock cycles for different combinations of both parameters and number of iterations. It is observed that the performance improves with the increase of the memory partition factor. It is found that the configuration that provides the best performance and that can be implemented in the ZedBoard is with 4 divisions of B2 and 32 partitions of memory. For this configuration we obtain an acceleration of approximately 209.83× in relation to the base architecture and 6.76× in relation to the CPU used as reference. The power consumption with this configuration is approximately 3.6 watts. For the Ultra96 the A3 architecture is implemented with a configuration of 4 divisions of B2 and 64 memory partitions. In this case an acceleration of 9.2× to 100 MHz and 18.21× to 200 MHz is achieved in relation to the sequential execution on CPU.

Acknowledgements. This study were supported by the AE&CC research Group COL0053581, at the Sistemas de Control y Robótica Laboratory, attached to the Instituto Tecnológico Metropolitano. This work is part of the project "Improvement of visual perception in humanoid robots for objects recognition in natural environments using Deep Learning" with ID P17224, co-funded by the Instituto Tecnológico Metropolitano and Universidad de Antioquia.

References

1. Castano, L., Osorio, G.: An approach to the numerical solution of one-dimensional heat equation on SoC FPGA. Revista Científica de Ingeniería Electrónica, Automática y Comunicaciones **38**(2), 83–93 (2017). ISSN 1815–5928
2. Cattaneo, R., Natale, G., Sicignano, C., Sciuto, D., Santambrogio, M.D.: On how to accelerate iterative stencil loops: a scalable streaming-based approach. ACM Trans. Archit. Code Optim. (TACO) **12**(4), 53 (2016)
3. Chugh, N., Vasista, V., Purini, S., Bondhugula, U.: A DSL compiler for accelerating image processing pipelines on FPGAs. In: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), pp. 327–338. IEEE (2016)
4. Cong, J., Li, P., Xiao, B., Zhang, P.: An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM (2014)
5. Deest, G., Estibals, N., Yuki, T., Derrien, S., Rajopadhye, S.: Towards scalable and efficient FPGA stencil accelerators. In: IMPACT 2016 - 6th International Workshop on Polyhedral Compilation Techniques, Held with HIPEAC 2016 (2016)
6. Deest, G., Yuki, T., Rajopadhye, S., Derrien, S.: One size does not fit all: implementation trade-offs for iterative stencil computations on FPGAs. In: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–8. IEEE (2017)
7. Del Sozzo, E., Baghdadi, R., Amarasinghe, S., Santambrogio, M.D.: A common backend for hardware acceleration on FPGA. In: 2017 IEEE International Conference on Computer Design (ICCD), pp. 427–430. IEEE (2017)
8. Escobedo, J., Lin, M.: Graph-theoretically optimal memory banking for stencil-based computing kernels. In: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 199–208. ACM (2018)
9. de Fine Licht, J., Blott, M., Hoefler, T.: Designing scalable FPGA architectures using high-level synthesis. In: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2018), vol. 53, pp. 403–404. ACM (2018)
10. Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y., Boku, T.: OpenCL-ready high speed FPGA network for reconfigurable high performance computing. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, pp. 192–201. ACM (2018)
11. László, E., Nagy, Z., Giles, M.B., Reguly, I., Appleyard, J., Szolgay, P.: Analysis of parallel processor architectures for the solution of the Black-Scholes PDE. In: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1977–1980. IEEE (2015)
12. Liu, J., Bayliss, S., Constantinides, G.A.: Offline synthesis of online dependence testing: parametric loop pipelining for HLS. In: 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 159–162. IEEE (2015)
13. Liu, J., Wickerson, J., Bayliss, S., Constantinides, G.A.: Polyhedral-baseddynamic loop pipelining for high-level synthesis. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **37**, 1802–1815 (2017)

14. Liu, J., Wickerson, J., Constantinides, G.A.: Loop splitting for efficient pipelining in high-level synthesis. In: 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 72–79. IEEE (2016)
15. Mokhov, A., et al.: Language and hardware acceleration backend for graph processing. In: 2017 Forum on Specification and Design Languages (FDL), pp. 1–7. IEEE (2017)
16. Mondigo, A., Ueno, T., Tanaka, D., Sano, K., Yamamoto, S.: Design and scalability analysis of bandwidth-compressed stream computing with multiple FPGAs. In: 2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), pp. 1–8. IEEE (2017)
17. Nacci, A.A., Rana, V., Bruschi, F., Sciuto, D., Beretta, I., Atienza, D.: A high-level synthesis flow for the implementation of iterative stencil loop algorithms on FPGA devices. In: Proceedings of the 50th Annual Design Automation Conference, p. 52. ACM (2013)
18. Natale, G., Stramondo, G., Bressana, P., Cattaneo, R., Sciuto, D., Santambrogio, M.D.: A polyhedral model-based framework for dataflow implementation on FPGA devices of iterative stencil loops. In: 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8. IEEE (2016)
19. de Oliveira, C.B., Cardoso, J.M., Marques, E.: High-level synthesis from C vs. a DSL-based approach. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, pp. 257–262. IEEE (2014)
20. Reagen, B., Adolf, R., Shao, Y.S., Wei, G.Y., Brooks, D.: Machsuite: benchmarks for accelerator design and customized architectures. In: 2014 IEEE International Symposium on Workload Characterization (IISWC), pp. 110–119. IEEE (2014)
21. Reiche, O., Özkan, M.A., Hannig, F., Teich, J., Schmid, M.: Loop parallelization techniques for FPGA accelerator synthesis. J. Signal Process. Syst. **90**(1), 3–27 (2018)
22. Sakai, R., Sugimoto, N., Miyajima, T., Fujita, N., Amano, H.: Acceleration of full-pic simulation on a CPU-FPGA tightly coupled environment. In: 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 8–14. IEEE (2016)
23. Sano, K., Hatsuda, Y., Yamamoto, S.: Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth. IEEE Trans. Parallel Distrib. Syst. **25**(3), 695–705 (2014)
24. Schmid, M., Reiche, O., Schmitt, C., Hannig, F., Teich, J.: Code generation for high-level synthesis of multiresolution applications on FPGAs. arXiv preprint arXiv:1408.4721 (2014)
25. Shao, Y.S., Reagen, B., Wei, G.Y., Brooks, D.: Aladdin: a pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In: ACM SIGARCH Computer Architecture News, vol. 42, pp. 97–108. IEEE Press (2014)
26. Waidyasooriya, H.M., Takei, Y., Tatsumi, S., Hariyama, M.: Opencl-based FPGA-platform for stencil computation and its optimization methodology. IEEE Trans. Parallel Distrib. Syst. **28**(5), 1390–1402 (2017)
27. Wang, S., Liang, Y.: A comprehensive framework for synthesizing stencil algorithms on FPGAs using OpenCL model. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2017)
28. Zohouri, H.R., Podobas, A., Matsuoka, S.: Combined spatial and temporal blocking for high-performance stencil computation on FPGAs using OpenCL. In: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 153–162. ACM (2018)