Optimizing Streaming Stencil Time-step Designs via FPGA Floorplanning

Marco Rabozzi, Giuseppe Natale, Biagio Festa, Antonio Miele, Marco D. Santambrogio

Politecnico di Milano, Milan, Italy
{marco.rabozzi, giuseppe.natale, antonio.miele, marco.santambrogio}@polimi.it, biagio.festa@mail.polimi.it

***Abstract*—Stencil computations represent a highly recurrent class of algorithms in various high performance computing scenarios. The Streaming Stencil Time-step (SST) architecture is a recent implementation of stencil computations on Field Programmable Gate Array (FPGA). In this paper, we propose an automated framework for SST-based architectures capable of achieving the maximum performance level for a given FPGA device through 1) the maximization of basic modules instantiated in the design and 2) optimization of the design floorplanning. Experimental results show that the proposed approach reduces the design time up to 15x w.r.t. naive design space exploration approaches, and improves the performance of the 13%.**

***Index Terms*—Field Programmable Gate Arrays, Floorplanning, Stencil Computations**

I. Introduction

Iterative Stencil Loops (ISLs) represent a class of algorithm that are highly recurrent in many High Performance Computing (HPC) applications such as differential equation solving [1] and scientific simulations [2]. The implementation of ISLs has been widely investigated in the literature, and their optimization has been approached in a variety of ways and targeting various architectures such as multi-core Central Processing Units (CPUs) [3], Graphic Processing Units (GPUs) [4] and Field Programmable Gate Array (FPGA) devices [5]. The ISLs regular structure allows them to be modeled via a powerful mathematical framework, called *polyhedral model* [6]. This model has been employed to either optimize ISL implementation on the target architecture, such as in [3], or design custom HW architecture tailored to this class of algorithms [5].

Among the available solutions, FPGA-based implementations (such as [5], [7], [8]) currently represent very promising candidates for ISL acceleration, as they offer a compelling trade-off between performance and power consumption thanks to direct hardware acceleration, while retaining flexibility achieved by means of their reconfigurability. One of the most interesting approaches in this scenario has been presented in [5]. The work proposes a design methodology for the automated generation of HW accelerators targeting FPGA devices starting from the C algorithmic description and exploiting commercial High Level Synthesis (HLS) tools. The obtained design consists of a highly repetitive and modular pipeline of a basic module, called Streaming Stencil Time-step (SST), that results in an efficient resource usage and scalability. However, such methodology lacks a back-end devoted to the implementation and optimization of the pipeline design onto the FPGA resource grid. In fact, since the SST-based architecture usually contains up to 100 modules, the absence of specific actions in the implementation flow leads to suboptimal performance and long design times.

This paper proposes an automated framework supporting the designer in the implementation of SST-based architectures. The framework receives as input the basic architectural template of the system from the methodology in [5] and maximizes the throughput of the implemented design by 1) preliminarily estimating the number of SST modules instantiable on the target FPGA device, and 2) floorplanning the design by means of a custom strategy to maximize the achievable clock frequency. As experimentally demonstrated, the main advantages of the proposed methodology are 1) a drastic reduction of the design time, since the approach is not based on iterative improvements, and 2) the possibility to increase the performance of the final design thanks to the exploitation of its characteristic regularity.

II. Background and Motivations

a) SST architecture: The basic structure of ISLs is depicted in Algorithm 1; the outer loop iterates for a given number of times, so called *time-steps*, while, at each time-step, the inner loop updates each value of the n-dimensional input vector by means of the *stencil* function, computing a weighted sum of the neighbor values in the vector.

Algorithm 1 Generic ISL Algorithm

```
for t ≤ TimeSteps do
    for all points p in matrix M do
        p ← stencil(p)
```

The architectural template of the SST-based accelerator [5] is depicted in Figure 1. The basic SST module performs the computation of a single time-step and is conceptually separated in a *memory system*, responsible for storage and data movement, and a *computing system*, that performs the actual computation. The SST module operates in a streaming mode on the various elements of the input vector; this internal structure is derived by means of a *polyhedral analysis* that allows to re-factor the algorithm to optimize on-chip memory resource consumption and implement a dataflow model of computation. Then, the complete SST-based architecture is obtained by replicating N times the basic module to implement a pipeline, where each module computes in streaming a single time-step of the outer loop of the algorithm. Such a pipeline is finally connected with a fixed communication subsystem.

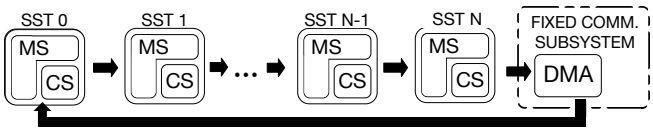
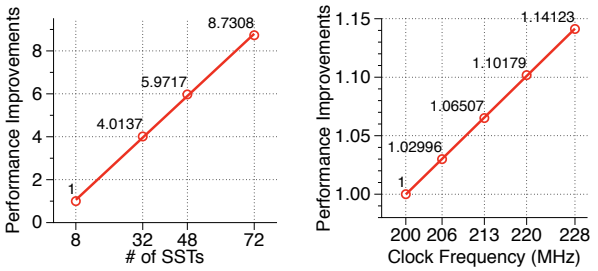


Fig. 1. The architectural template of the SST-based accelerator for ISL.

b) SST performance evaluation: As discussed in [5], the performance of such architecture scales almost linearly with the increase in the number of SSTs (Figure 2a). Indeed, as the overall number of iteration of an ISL algorithm is usually very large, performance can be improved by maximizing the number



(a) # instantiated SSTs (b) Frequency

Fig. 2. Experimentally-measured performance scaling for Jacobi2D algorithm.

of modules in the pipeline within the limits of the available device resources. The result is that the accelerator implements a higher fraction of the total number of time-steps of the ISL, reducing the number of needed sweeps through the accelerator. Within this paper we also analyzed the impact of the clock frequency targeted in the implementation on the overall system performance. As an example, Figure 2b reports the performance speedup of a Jacobi2D algorithm featuring a single SST module, synthesized on a Xilinx Virtex xc7vx485 device [9] by means of Xilinx Vivado. We notice that also frequency improvements result in a linear increase in performance. Hence, from an analytical point of view, the performance of this architecture can be estimated as:

$$T_q = \frac{T_0}{q} \cdot \frac{f_0}{f_q} \tag{1}$$

where T_q is the completion time of the system with q instantiated SSTs implemented at frequency f_q and T_0 is the one of a system with a single module implemented at frequency f_0.

The original methodology proposed in [5] explores different numbers of SSTs instantiated in the pipeline with a simple trial-and-error flow based on resource estimation. The approach does not take frequency into account and performs several synthesis to find the actual number of SSTs, thus requiring a considerable design time. In this paper we define a novel framework acting as a back-end for the methodology in [5]. As we will show in the experimental session, we can provide a more accurate estimation of the resources based on a preliminary floorplanning of the design. Nevertheless, a specific floorplanning activity may also allow to reach final implementations guaranteeing the timing closure at clock frequencies not reachable by an unconstrained implementation.

III. Proposed Approach

The objective of the framework is twofold: first, maximize the performance of the overall system by determining the maximum number of SSTs that can be instantiated onto the FPGA and by allowing to increase the operational frequency; second, reduce the overall synthesis and implementation time that would be incurred with simple iterative design approaches.

We leverage floorplanning for our purpose. This allows to increase the frequency of the system and the number of SSTs that can be instantiated while gaining timing closure. Moreover, floorplanning allows to directly determine the number of SSTs that can be instantiated without running multiple implementation attempts thus requiring significantly long design times.

The overall framework is internally organized in three phases, and is interfaced with a commercial synthesis and implementation tool (Xilinx Vivado has been here used). In *Phase 1*, we consider a basic version of the design consisting in a single SST accelerator and the fixed communication subsystem. Such design is synthesized in order to obtain a first SST resource estimation; the module is further refined by floorplanning the design and validating the place and route results against several possible choices in the size and shape of the placement region. Once the minimal SST region size is determined, *Phase 2* leverages an Integer Linear Programming (ILP) model to maximize the number of SST regions that can be floorplanned and solves an euclidean Traveling Salesman Problem (TSP) to find an optimal interconnection order for the SST accelerators. Finally, *Phase 3* implements the full design with the identified floorplanning constraints and runs multiple synthesis to determine the maximum design frequency.

Phase 1 and 2 require a preliminary generation of a set of placements covering a given amount of FPGA resources. Hence, we first present this preliminary placements generation process and the FPGA characterization. Then, the details of the three framework phases are discussed.

A. FPGA model and placements generation

Similarly to most of the floorplanning algorithms available in the literature [10], [11], we model an FPGA device as a matrix of tiles, each one containing a single type of resource such as Configurable Logic Blocks (CLBs), Digital Signal Processors (DSPs) and Block RAMs (BRAMs).

Since, in our scenario we are addressing only static designs, in order to maximize the overall utilization of the device, we take into account a fine grain tile that spans a single resource wide, while its height covers the minimum integral number of resources for each resource type (e.g. either 2 DSPs, 2 BRAMs or 5 CLBs for Virtex7 devices). We denote with W and H the number of tiles on the horizontal and vertical directions respectively ($W = 151$ and $H = 70$ for a Virtex xc7vx485). A placement $p = (x, y, w, h)$ on the tiles grid is defined as a rectangular shape starting in the bottom-left corner (x, y) and spanning w tiles wide and h tiles height. Furthermore, we denote with $r_p = (n_{CLB}, n_{DSP}, n_{BRAM})$ the resource vector of placement p that specifies the number of CLB, DSP and BRAM covered by the placement. Upon the presented FPGA model, we define a utility routine leveraged by Phases 1 and 2 of our framework that is in charge of the generation of the set of all the possible placements for a single SST. In our formulation, a valid placement is defined by specifying a width w such that: 1) the region has the minimum width to form a bounding box covering all the resources required by the SST module, 2) it presents a given aspect ratio (i.e. the width/height ratio). Algorithm 2 shows the procedure for SST placements generation; the function *searchWidth* uses binary search to find the minimum width in the range $[minW, maxW]$ compatible with the aspect ratio constraint. If *searchWidth* is unable to find such value, or the identified placement overlap with a hardware macro or forbidden user-defined area, the function returns 0. Finally, we assume that the placement of the fixed SST communication subsystem is predefined and no placements overlapping with such region are generated.

B. SST resource requirements analysis

Phase 1 consists in determining the minimum resource requirements for a placement hosting an SST module. The first performed step is the synthesis of a simple design containing only a single SST and the communication subsystem. Thus, we collect the estimated SST resource vector $\hat{r}_{SST} =$

Algorithm 2 SST placements generation

```
1: a ← maximum aspect ratio (≤ 1)
2: r ← resource requirement vector
3: P ← ∅
4: for x ← 0 to W − 1 do
5:     for y ← 0 to H − 1 do
6:         for h ← 1 to H − y do
7:             minW ← a · h
8:             maxW ← (1/a) · h
9:             w ← searchWidth(x, y, h, minW, maxW, r)
10:            if w > 0 then
11:                P ← P ∪ (x, y, w, h)
```

$(\hat{n}_{CLB}, \hat{n}_{DSP}, \hat{n}_{BRAM})$ and generate the set of placements $\hat{P}$ that covers the resource vector $\hat{r}$. Among the identified placements, we select the one that covers the least amount of CLB resources and do not overlap with the predefined area for the fixed components. Finally, we perform the place and route of the design using a runtime optimized strategy (such as the *Flow Quick* implementation strategy available in Xilinx Vivado) that limits the overall implementation time. Depending on the success or failure of the place and route, we repeat the entire procedure by respectively decreasing or increasing $\hat{n}_{CLB}$ by $\Delta = 5$ CLB tiles (corresponding to half a CLB tile on Virtex7 devices). The algorithm terminates once the minimal placement size for which the place and route succeed is determined.

C. Maximal floorplan generation

In Phase 2, based on the resource requirements derived from the previous phase, we simultaneously identify the maximum number of SSTs that can be placed into the device and their floorplanning constraints. More formally, given a resource vector r for a single SST and the set of placements P generated using Algorithm 2, the objective is to find a maximal subset of placements $F \subseteq P$ such that no two distinct placements $p_1, p_2 \in F$ overlap. This problem is quite different from the one addressed by classical FPGA floorplanning algorithms such as [10]–[12]; indeed in our scenario we are not given a specific set of regions to floorplan, but we need to maximize their number. In order to identify such a maximal floorplan, we propose an ILP-based strategy that allows to find the optimal number of placements within a limited amount of time. For every placement $p \in P$, we associate a binary variable x_p that is set to 1 if and only if the corresponding placement p is selected in the final solution. Hence the ILP objective is:

$$max \sum_{p \in P} x_p \tag{2}$$

Moreover, a feasible solution requires no overlapping among the selected placements. As described in [10], such constraint can be translated into requiring that at most one placement is selected out of those covering a specific FPGA tile:

$$\forall\ xt \in \{0, 1, \ldots, W-1\},\ yt \in \{0, 1, \ldots, H-1\} : \sum_{\substack{p=(x,y,w,h) \in P\ \mid \\ x \leq xt < x+w, \\ y \leq yt < y+h}} x_p \leq 1 \tag{3}$$

Once the ILP model is solved, the maximal subset of placements is $F = \{p \in P \mid x_p = 1\}$. At this stage of the phase we have identified the number $|F|$ of SSTs and their floorplanning constraints, but we have not yet taken into account the modules interconnections. To optimize the interconnections among the SSTs, we exploit the regular design structure shown in Figure 1. Indeed, it is easy to note that the interconnection topology of the SSTs and the fixed subsystem form a ring. Furthermore, since the SSTs are all replicas of the same basic module, we are allowed to interchange the connections of two elements in the ring topology without affecting the functionality of the design. Thanks to these properties, the problem of optimizing the SST interconnections translates into finding a minimum TSP tour that visits the centers of each SST placement and the fixed part of the design exactly once. For the TSP optimization, we consider the euclidean distance among the centroids of the placements, as it ensures no overlap among the interconnections, and we leverage an exact TSP solver to find the optimal solution.

D. System implementation and frequency scaling

In Phase 3, the framework explores different target frequencies to determine the maximum one that allows timing closure during the system place and route. The exploration is performed within an interval $I = [f_{min}, f_{max}]$ by first implementing a design at frequency f_{min} to verify the existence of a feasible design, and subsequently exploring with binary search other target frequencies in the interval. Due to technological constraints, the actual frequency values that can be selected in the interval I consist in a small set of frequency choices $f_0, f_1, \cdots, f_n$. Hence, the binary search terminates returning frequency f_i as soon as it verifies that timing closure is met at frequency f_i but not at f_{i+1} or if $f_i = f_{max}$. Compared to the SST maximization, frequency exploration is a time consuming process since each binary search iteration requires to re-synthesize the overall design. Nevertheless the same design time overhead would also apply for the approach discussed in [5] since no explicit frequency estimation approach is defined.

IV. Experimental Evaluation

The framework has been evaluated on three ISL computations (Jacobi2D, Heat3D and Seidel2D) targeting a Xilinx Virtex xc7vx485 device [9]. The synthesis and implementation have been performed using Xilinx Vivado 2015.4, the ILP models have been solved by using Gurobi 7.0.2 [13], and the TSP problem by means of the exact Concorde TSP solver [14]. Experiments have been performed on a Intel Core i7-6700 CPU at 3.40 GHz with 32 GBs of RAM. A maximum aspect ratio of 2.5 has been used for Algorithm 2, while we used the range $[160MHz, 250MHz]$ for frequency exploration.

In our experimental campaign we compared the Proposed Approach (PA) against the baseline design methodology defined in [5], that for the sake of fairness in the comparison, has been enhanced with the final frequency exploration strategy defined in Section III. Table I reports the achieved performance and execution time of both approaches. As can be noted, thanks to FPGA floorplanning we are able to increase the target frequency for the Jacobi2D and Heat3D algorithms of approximately 11%. Indeed, since the total number of SST that can be placed into the design is small for this benchmark due to the internal complexity and resource requirement of the single SST module, the floorplanning reduces its impact on the overall design by leaving more room to the place and route algorithm. Additionally, the Seidel2D base SST is intrinsically more complex, as this ISL has *spatial dependencies* between point updates, and thus updates within a time-step are inherently sequential [5]. Due to this characteristics, the resulting SST design is more convoluted and more eager of logic resources. As can be noted, the maximal floorplanning algorithm allows

TABLE I
ANALYSIS OF THE DESIGN SPACE EXPLORATION (DSE) EXECUTION TIME AND ACHIEVED SYSTEM PERFORMANCE.

Algorithm	SSTs maximization time (# synthesis runs)		frequency exploration time (# synthesis runs)		PA overall time reduction w.r.t. [5]	# SSTs		design frequency (MHz)		PA performance improvement w.r.t. [5]
	PA	[5]	PA	[5]		PA	[5]	PA	[5]	
Jacobi2D	6.36h (1)	382.00h (40)	19.30h (3)	24.30h (4)	15.84x	90	88	228	206	13.20%
Heat3D	4.19h (1)	13.91h (2)	14.66h (3)	17.94h (4)	1.69x	25	25	228	206	10.68%
Seidel2D	4.95h (1)	8.60h (2)	17.31h (4)	15.42h (4)	1.08x	19	19	183	183	0%

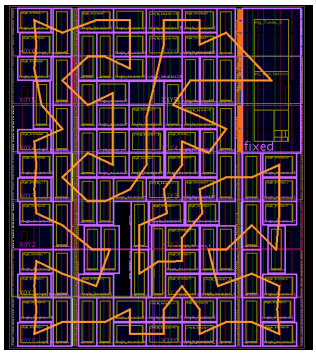


Fig. 3. Floorplan achieved by the proposed approach on the Jacobi2D algorithm. The fixed communication components are placed on the top-right corner, while the other regions constrain the position of 90 SST modules.

TABLE II
COMPARISON OF THE PROPOSED APPROACH AND THE ITERATIVE VERSION OF PRFLOOR [11] USING HALF-CLOCK REGION TILES.

Algorithm	# SSTs		execution time	
	PA	PRFloor	PA	PRFloor
Jacobi2D	90	64	0.18s	4207.39s
Heat3D	20	18	1.77s	450.41s
Seidel2D	18	15	0.78s	209.97s

to greatly reduce the number of required synthesis within the SSTs maximization stage, thus leading to an execution time saving of 15.84x for Jacobi2D. Instead, the execution time of the frequency exploration phase does not change significantly among the two approaches as it only depends on the binary search iterations.

Finally, we aimed at demonstrating also the higher efficiency of our strategy specifically targeted to the SST design w.r.t. the employment of one of the standard FPGA floorplanners proposed in the literature. To this purpose we adopted *PRFloor*, i.e. the most recent full-fledged strategy presented in [11][1] and used it in an iterative fashion by floorplanning an increasing number of SST modules until a non feasible solution is found. In order to align to the PRFloor algorithm, we here consider tiles spanning half a clock region. Furthermore, since the final trial-and-error placement procedure of PRFloor is not completely defined in [11], we adopted instead an exact ILP model equivalent to the one discussed in [10], but targeting only the subset of placements that are close to the anchor points identified by the PRFloor bipartitioning phase. As a consequence, the modified PRFloor is guaranteed to find a floorplan solution, if it exists, even if it might incur in a higher running time. Table II compares the number of SSTs identified by the algorithms as well as their running time. Notice that since we increased the granularity of the tiles, the numbers of SSTs found by our approach can be smaller than the ones presented in the previous experimental session. As it can be noticed from the table, PRFloor is not able to identify the maximum number of SST even if uses a considerable amount of time. The cause is that the bipartitioner does not fit well the regular ring topology of the SST-based accelerator. Indeed, a ring topology leads to several optimal partitioning solutions, however, since PRFloor considers the horizontal and vertical cuts independently, it has a low probability of spreading the modules evenly on the FPGA.

[1]It is worth mentioning that we reimplemented the algorithm in [11] since it is not publicly available.

V. CONCLUSIONS

This paper proposed an automated back-end framework for SST-based architectures for the design methodology in [5]. Experimental results have demonstrated the capability of the proposed approach to reduce the design time up to 15x w.r.t. naive design space exploration approaches, and, at the same time, to improve the performance of the 13%.

ACKNOWLEDGMENTS

The work has been partially funded by EXTRA EU project (EU Horizon 2020 program, grant No 671653).

REFERENCES

[1] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey, "A finite-volume, incompressible Navier-Stokes model for studies of the ocean on parallel computers," *Journal of Geophysical Research*, vol. 102, no. C3, pp. 5733–5752, Mar. 1997.
[2] A. Nakano, R. K. Kalia, and P. Vashishta, "Multiresolution Molecular Dynamics Algorithm for Realistic Materials Modeling on Parallel Computers," *Computer Physics Comm.*, vol. 83, no. 2, pp. 197–214, 1994.
[3] V. Bandishti, I. Pananilath, and U. Bondhugula, "Tiling Stencil Computations to Maximize Parallelism," in *Proc. Intl. Conf. on High Performance Comp., Networking, Storage and Analysis*, 2012, pp. 40:1–40:11.
[4] J. Holewinski, L.-N. Pouchet, and P. Sadayappan, "High-performance Code Generation for Stencil Computations on GPU Architectures," in *Proc. of the Intl. Conf. on Supercomputing*, 2012, pp. 311–320.
[5] G. Natale, G. Stramondo, P. Bressana, R. Cattaneo, D. Sciuto, and M. D. Santambrogio, "A Polyhedral Model-based Framework for Dataflow Implementation on FPGA Devices of Iterative Stencil Loops," in *Proc. of Intl. Conf. on Computer-Aided Design (ICCAD)*, 2016, pp. 77:1–77:8.
[6] P. Feautrier and C. Lengauer, "Polyhedron model," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1581–1592.
[7] A. A. Nacci, V. Rana, F. Bruschi, D. Sciuto, P. di Milano, I. Beretta, and D. Atienza, "A high-level synthesis flow for the implementation of iterative stencil loop algorithms on FPGA devices," in *Proc. of Design Automation Conference (DAC)*, 2013, pp. 1–6.
[8] K. Sano, Y. Hatsuda, and S. Yamamoto, "Scalable Streaming-Array of Simple Soft-Processors for Stencil Computations with Constant Memory-Bandwidth," in *Proc. of Intl. Symposium on Field-Programmable Custom Computing Machines*, 2011, pp. 234–241.
[9] Xilinx Inc., http://www.xilinx.com.
[10] M. Rabozzi, J. Lillis, and M. D. Santambrogio, "Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming," in *Proc. Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 186–193.
[11] T. D. A. Nguyen and A. Kumar, "PRFloor: An Automatic Floorplanner for Partially Reconfigurable FPGA Systems," in *Proc. of Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*, 2016, pp. 149–158.
[12] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *Proc. Intl. Conf. on Reconfigurable Computing (ARC)*, 2012, pp. 13–25.
[13] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com
[14] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Concorde TSP solver," 2006. [Online]. Available: http://www.tsp.gatech.edu/concorde