RESHAPE: A Run-time Dataflow Hardware-based Mapping for CGRA Overlays

Maria Vieira*, Michael Canesche*, Lucas Bragança*, Josué Campos*, Mateus Silva*,
Ricardo Ferreira*, Jose A. Nacif*
E-mail: jnacif@ufv.br
* Universidade Federal de Viçosa, Viçosa, Cep 36570 900, Brazil

***Abstract*—Coarse-grained reconfigurable architectures (CGRA) are a power-efficient approach for hardware accelerators. However, there are few EDA tools for CGRA. We develop hardware-based placement and routing (P&R) for fully-pipelined CGRA mapped as an FPGA overlay. The key idea is to use the available FPGA resources to replicate several mapping units, thus exploring parallel execution, area/execution time trade-offs, and achieving near-optimal mapping solutions. Furthermore, our P&R provides portability and an incremental run-time approach. In comparison to VPR and CGRA-ME tools and a time-multiplexer approach, our spatial mapping reduces the P&R execution time, and it improves the performance up to hundreds of Gops/s by using fully-pipelined architectures.**

I. Introduction

Overlays of coarse-grained reconfigurable architectures (CGRAs) for FPGA are widely adopted to mitigate the lack of flexibility of ASICs and the time complexity of the process of compiling a bitstream for an FPGA [1], [2]. CGRAs ensure flexibility by accelerating loop iterations for signal-processing, multimedia, Internet-of-Things, and bioinformatics [3]–[7]. However, mapping an application for CGRA is an NP-complete problem [8]. Currently, software-based development tools are scarce and may require minutes to hours to find a good mapping [9]. Several heuristics to solve these problems have been presented based on simulated annealing [10]–[12], SAT-solver [13], greedy approach [14], [15], genetic algorithm [16], integer linear programming [9], split-and-merge [17], and deep learning [18], [19].

In contrast to recent work [20], [21] on time-multiplexer FPGA overlays, in which the performance is limited to a few Gops/s, we present a spatially configured overlay CGRA (SC). In our CGRA, we implement the fully-pipelined model achieving a theoretical performance of 395.3 GOPs/s. It is also possible to attach our CGRA to a high throughput accelerator interface, as proposed in [22]. Furthermore, our approach maximizes the FPGA occupation, therefore avoiding wasting resources that would be left unused. By using data-flow and mapping unit replication, we improve both mapping time and CGRA performance. Our main contributions are (1) a run-time hardware-based CGRA mapping; (2) a new FPGA unused area allocation strategy to perform incremental and on-the-fly CGRA fine-tuning mapping ; (3) a high-performance CGRA overlay spatially configured as a fully-pipelined architecture.

This paper is structured as follows: Section II presents a CGRA overview and describes RESHAPE, our incremental hardware-based placement, and routing. Section III presents and discusses the results. Finally, Section IV closes our remarks and proposes future work.

II. RESHAPE - Architecture-Independent Mapping

A generic CGRA is composed of a set of processing elements (PEs) and an interconnection network. Figure 1(a) and Figure 1(c) respectively show mesh and mesh-plus interconnections. Figure 1(b) depicts the PE structure, which has a functional unit (FU), multiplexers, and registers. Furthermore, the PE has a configuration memory for local interconnections (routing) and FU operation (placement).

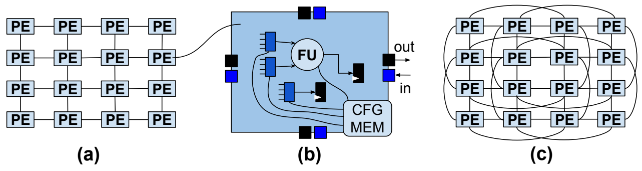


Fig. 1. (a) CGRA; (b) PE plus Routing Register; (c) MESH PLUS.

This work focuses on spatially mapping a data-flow onto the CGRA. This approach is named spatial compute (SC) [9]. Therefore, we use simple PEs, which are not shared temporarily or multiplexed every clock cycle [8], [21], named time-multiplexer (TM) approaches. Although CGRA with temporally shared PEs saves hardware resources, on the other hand, the mechanisms for changing the context of operation in the PE and the algorithms for mapping and routing require more complex algorithms such as module scheduling [5], [23].

The spatial CGRA mapping is challenging because we should balance all pipeline paths by inserting queues after mapping. The number of inserted queues can significantly impact architecture cost and throughput [24]. In this work, we also evaluate an asynchronous CGRA model, where unbalanced paths do not require registers, although throughput degradation may occur, as is showed in subsection II-D.

We depict a general view of our proposal in Figure 2. A front-end step generates the data-flow graph from high-level code. The data-flow is sent to the mapping unit inside the FPGA①. We implement the CGRA and the mapping unit as an FPGA overlay. Therefore, we provide portability at the bitstream level by isolating the front-end. Our mapping input is a simple high-level data-flow. Next, the mapping unit performs the placement and routing ②. Then, the generated bitstream is uploaded ③, and, finally, the CGRA starts the execution by receiving/sending from/to data ④.

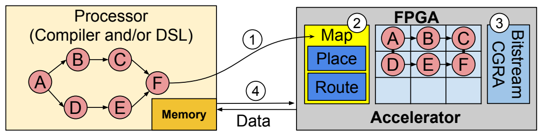


Fig. 2. CGRA Mapping and Execution Framework.

A. Placement

CGRA mapping consists of two steps: Placement and Routing (P&R). We simplify a greedy placement proposed in [25] to develop an FSM-based implementation in hardware. First, we represent the graph as a spanning-tree edge list (STEL) plus a list of remained edges (RE). The STEL has an additional property where each node appears once as the edge destination. Assume a graph (V, E) where V is the set of nodes and E the edges. Suppose $V = \{x, y, z\}$ and $E = \{x \to y, y \to z, x \to z\}$. Suppose the STEL=$\{x \to y, y \to z\}$ and the RE=$\{x \to z\}$. The algorithm scans the STEL, and for each edge $A \to B$, the node B is placed as a neighbor of A. In our example, we randomly place the node x, then the edges $\{x \to y, y \to z\}$ are processed, where x determines the position of y, and y determines the position of z, as shown in Figure 3(a) (blue color). Therefore, the greedy approach transfers the graph locality to the CGRA locality. However, it does not consider the edges that belong to RE during the placement. We route these edges after the placement step with no guarantee regarding the RE routing locality. In our example, we route the edge $x \to z$ by using a bypassing PE, as shown in Figure 3(a) (red color). Finally, we can summarize as follows: 1) STEL scanning is the placement and local routing steps; 2) RE scanning is the final routing step.

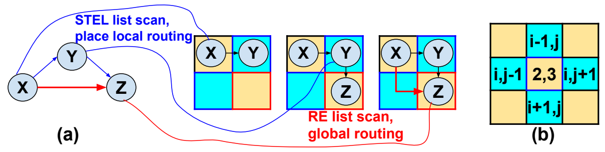
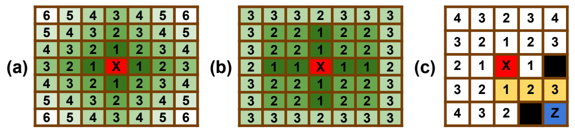


Fig. 3. (a) STEL and RE list; (b) *GetPosition* in the neighborhood of $PE_{2,3}$

Figure 3(a) shows in blue color the free nodes when the algorithm process the edges $x \to y, y \to z$ to place y and z, respectively. Assuming A placed at $PE_{i,j}$, for instance $PE_{2,3}$, as shown in Figure 3(b), the *GetPosition* looks-up at adjacent relative free positions: $(i, j+1), (i, j-1), (i-1, j), (i+1, j)$. The *offset* vector stores these relative indexes in a simple and generic way, i.e. $(0, +1), (0, -1), (-1, 0), (+1, 0)$. We add the current PE to the *offset* vector to check for a free position. We also implement a random shuffle in the *offset* vector to spread out the placement across the grid. When there is no free node in adjacent neighborhood, our algorithm performs a breadth-first search to find a free position.

B. Routing

We propose an FSM-based simplified version of the maze routing algorithm to establish the remaining edges. Our implementation does not perform backtracking, and we restrict the path-search to a single one-shot approach. The path-search strategy is based on the Breadth-First Search (BFS) by using Manhattan distance between origin and destination, as shown in Figure 4(a) for mesh and Figure 4(b) for mesh-plus. Figure 4(c) shows a routing example for the edge $x \to z$, where the black squares are obstacles, and the yellow path represents the solution given by our algorithm.



6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
3	2	1	X	1	2	3
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6

3	3	3	2	3	3	3
3	2	2	1	2	2	3
3	2	2	1	2	2	3
2	1	1	X	1	1	2
3	2	2	1	2	2	3
3	2	2	1	2	2	3
3	3	3	2	3	3	3

4	3	2	3	4
3	2	1	2	3
2	1	X	1	■
3	2	1	2	3
4	3	2	■	Z

Fig. 4. Manhattan distances from X: (a) Mesh; (b) Mesh-plus; (c) Routing $x \to z$.

C. A Run-time Incremental Approach

We propose an incremental approach. First, we implement our P&R units inside the FPGA, and it consumes few resources. We take advantage of the unused FPGA area to replicate the number of P&R units as much as possible. Therefore we can perform several P&R attempts in parallel. Second, it is possible to improve our P&R by multiple executions of the algorithm. Furthermore, once we successfully map the first solution, our P&R units can still execute in parallel to search for better mapping, replacing the current solution at run-time.

D. Asynchronous Data-Flow

One approach to avoid FIFOs is asynchronous data-flow mapping. A PE can process an operation if and only if all input data are available at the correct time frame [24], [26]. The throughput can be smaller than one. Consider the example depicted in Figure 5(a), where a long wire delays the edge $d \to c$ in two clock cycles. Figure 5(b) shows when c receives all input data, and one cycle later the first result is produced (see Figure 5(c)). However, although the following data is already available from b, it takes two clock cycles to traverse the long wire $d \to c$, as shown in Figures 5(c-f). Therefore, the throughput is one result at each three clock cycles, or $\frac{1}{3}$.

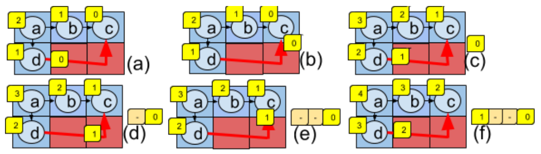


Fig. 5. (a) The first $data_0$ arrives at c; (b) Both $data_0$ are available at c; (c-f) Waiting and processing $data_1$ where the throughput is $\frac{1}{3}$.

Furthermore, as already mentioned, we can improve the throughput by finding better mapping, as shown in Figure 6. Therefore, while the throughput is not fully optimized, our mapping unit can work parallel to the CGRA execution, searching for better mapping solutions.

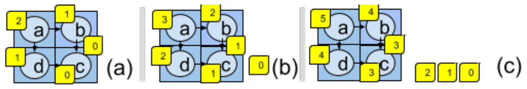


Fig. 6. (a) The $data_0$ arrives at c; (b) First result; (c) Throughput is 1.

III. Experimental Results

We organize this section as follows. In subsection III-A, we generate three CGRA overlays to validate our approach with different trade-offs among FIFO and routing resources. In subsection III-B, we evaluate the CGRA performance in GOPs/s as a function of P&R execution tries targeting a fully-pipelined architecture over a set of benchmarks from [21]. In subsection III-C, we evaluate the trade-offs targeting an asynchronous architecture. In subsection III-D, we measure the FPGA resources required to implement our P&R unit. In subsection III-E, we compare the execution time of the proposed hardware P&R to a software implementation on a high-performance multicore CPU. Finally, in subsection III-F, we compare the P&R quality and execution time to VPR [11], CGRA-ME [27], and the approach proposed in [21].

A. CGRA Overlay

We generated three CGRA overlays with mesh-plus topology to evaluate our approach. Each PE input has an elastic FIFO to balance the paths in a fully-pipelined implementation. The routing and the FIFOs use multiplexers and registers, which consume a more considerable amount of FPGA resources. The architecture A_1 offers full PE routing capability and an extensive input FIFO (size=8) to simplify the P&R task. However, the price to pay to benefit from its advantages is the CGRA cost, where the maximum size is $22 \times 22 = 484$ PEs in the target FPGA. The architecture $A_2 = 784$ increases the CGRA size to 28×28 by reducing the input FIFO size to 4, which increases the complexity of the timing phase after the routing. Finally, the architecture A_3 increases, even more, the CGRA size up to $36 \times 36 = 1,296$ by reducing the PE bypassing routing capability to a single connection. The A_3 clock frequency is 305 MHz, which can reach the peak performance of 395.3 GOPs/s. Table I shows three designs.

TABLE I
CGRA Overlay with Mesh-Plus, FIFOs and Routing Resources.

Arch	grid size	FIFO	Resources Route	ALM	DSM	clk MHz	Percentage % ALM	DSP
A_1	22x22	8	Full	307686	484	262	72.0	31.9
A_2	28x28	4	Full	405119	784	287	94.8	51.6
A_3	36x36	4	One	367237	1296	305	86.0	85.4

Our fully-pipelined approach focuses on high performance and FPGA occupation maximization compared to the time-multiplexer (TM) approach [21], which reduces the CGRA size, thus degrading performance. We believe that if there are free resources in the FPGA, we should maximize their usage. In this direction, we have extended the TM approach [21] by using graph replication to map as many as possible data-flow in an 8×8 CGRA overlay [20]. Nevertheless, in comparison to a 8×8 CGRA, the A_1, A_2, and A_3 present a peak speedup of 9×, 12.25×, and 20.25×, respectively.

B. Throughput in GOPs/s

This section evaluates the CGRA performance for a data-flow set from [21] in the three target architectures. We map a data-flow in a minimal square plus graph replication to maximize the performance and the CGRA usage. For instance, the *mibench* graph has 27 nodes and therefore, maps onto a $6 \times 6 = 36$ grid. If the target CGRA is the A_1 (see Table I) with 22×22 PEs, we can instantiate $3 \times 3 = 9$ data-flows copies without overlapping. The target architecture is chosen based on the routing results and required resources (FIFO size and single/full PE routing). Our *reshape* approach can improve the mapping at run-time. For instance, one single placement execution for the *mibench* requires FIFO size 4 and full routing, i.e. CGRA architecture A_2. When we continue to execute the mapping up to 100 instances, the best placement requires only FIFO size 1 and single routing PEs. It is also possible to replace the CGRA A_2 with the CGRA A_3, which increases the number of data-flow copies.

TABLE II
Throughput in GOPs/s, FIFO and target CGRA.

Bench	GOPs/s 1	100	1,000	FIFO and CGRA 1	100	1,000
chebyshev	173	173	173	1,A_3	0,A_3	0,A_3
mibench	58	143	143	4,A_2	1,A_3	0,A_3
poly5	-	121	206	-	2,A_2	1,A_3
poly6	-	53	53	-	6,A_1	6,A_1
poly8	38	156	156	8,A_1	4,A_3	3,A_3
qspline	-	122	122	-	2,A_3	2,A_3
sgfilter	81	198	198	3,A_2	1,A_3	1,A_3

Table II shows the results in GOPs/s for 1, 100, and 1,000 executions of our mapping algorithm. The last three columns show the FIFO size and the target CGRA. In comparison to previous work [20], [21] which achieve the 35 GOPs/s for 16 copies of *chebyshev* in 16×16 DSPs CGRA, we achieve 173 GOPs/s (4.94× more) in a 36×36 CGRA (5.06× higher). Therefore, we can keep the performance and scale up the architecture.

C. Asynchronous and Throughput

This section evaluates an asynchronous target architecture to reduce the required FIFO and the throughput of a fully-pipelined (FP) CGRA architecture. The FP CGRA throughput is always 100%, and it produces a new result every clock cycle. An asynchronous PE computes an operation if only if all input data have arrived. Table III shows maximal throughput reduction for 1, 100, and 1000 mapping executions. The 5^{th} to the 7^{th} columns show the total routing wire lengths. The last column (Opt) shows the optimal wire length solution. Although a better mapping reduces the wire length, it seems negligible from 100 to 1,000 instances. However, it can impact the final throughput. For instance, the sgfilter reaches the optimal throughput by reducing only one routing wire segment.

TABLE III
ASYNCHRONOUS THROUGHPUT IN PERCENTAGE %

	Throughput			Wire Length			
Bench	1	100	1,000	1	100	1,000	OPT
chebyshev	85.7	99.9	99.9	20	19	19	19
mibench	71.4	85.7	99.9	39	34	34	32
poly5	76.1	76.1	79.9	-	78	76	60
poly6	59.9	73.6	73.6	-	191	184	125
poly8	39.9	75.5	79.9	123	106	106	83
qspline	73.6	79.9	83.2	-	116	114	85
sgfilter	76.8	88.8	99.9	54	47	46	42

D. Place and Route Units: FPGA Resources

We propose to implement as many P&R units in the unused FPGA area to maximize even more the FPGA occupation and speedup the P&R by performing parallel execution. Table IV presents the FPGA resources and percentage of total resources for our hardware-based P&R, where we show: the area in Adaptive Logic Module (ALM), Total of Block Memory Bits (TBMB), and RAM Blocks (RAMB). We evaluate two dataflow grid sizes $9 \times 9 = 81$ and $11 \times 11 = 121$ PEs. The placement unit (PU) consumes only 0.13% of the ALM resources. Hence, 100 PU consumes 13% of FPGA ALMs. The routing unit consumes less than 0.16% of the FPGA area. The consumption of 100 P&R for both units occupies 29% of the FPGA.

TABLE IV
P&R UNTIS: ARIA10 FPGA RESOURCES

grid	Placement Unit			clk	Routing Unit			clk
size	alm	tbmb	ramb	MHz	alm	tbmb	ramb	MHz
	0.13%	0.01%	0.1%		0.16%	0.01%	0.03%	
81	565	5760	3	292	688	8192	1	348
	0.13%	0.01%	0.1%		0.19%	0.01%	0.03%	
121	568	5760	3	274	835	8192	1	336

E. P&R Execution Time

The P&R execution time from the previous work [20] requires 220ms using a Xeon E5 3.5 GHz and 880ms using the ARM 667 MHz embedded FPGA processor. Table V shows our hardware-based P&R execution time in comparison to a software version of our P&R for 1, 100, and 1,000 instances in a Ryzen 3 CPU 3.6 GHz. The average CPU execution time for 100 instances is 0.41ms, while the average for the hardware-based placement unit is 0.015ms, and the routing unit is 0.003ms. Therefore, our software or hardware P&R perform orders of magnitude faster than previous work on the same benchmark set [20]. Our hardware approach is twice faster than the software version. Furthermore, if we consider the A_3 architecture, it is possible to allocate 50 P&R units, and therefore, the hardware unit, on average, computes 100 copies in 60ms, which is $5\times$ faster than the 100 copies in CPU.

F. Previous Work

Table VI presents the mapping costs and execution times for mesh-plus architecture in comparison to previous work: 1) VPR [11] uses the well-known simulated annealing FPGA P&R; 2) CGRA-ME [27] uses integer linear programming (ILP) and does not scale to larger graphs. Our P&R approach achieves the same wire length results compared to VPR and is $37\times$ faster. We execute VPR [11] in the fast mode.

TABLE V
PLACEMENT AND ROUTING EXECUTION TIME IN MILLISECONDS.

	Node			CPU		
Bench	Size	Place	Route	1	100	1,000
chebyshev	16	0.005	0.002	0.017	0.091	6.041
mibench	27	0.007	0.002	0.016	0.145	9.594
poly5	47	0.015	0.005	0.052	0.247	21.404
poly6	101	0.027	0.004	0.085	1.193	80.827
poly8	68	0.018	0.004	0.044	0.447	43.425
qspline	69	0.022	0.004	0.049	0.517	40.879
sgfilter	36	0.012	0.004	0.037	0.248	14.704

TABLE VI
VPR × CGRA-ME × RESHAPE: MAPPING TIME IN MILLISECONDS

	VPR		CGRAME		RESHAPE	
Bench	time	Wire	time	Wire	time	Wire
chebyshev	2.71	24	$1.0 \cdot 10^5$	19	0.091	23
mibench	7.42	41	$1.8 \cdot 10^5$	32	0.145	41
poly5	13.95	83	$3.5 \cdot 10^5$	-	0.247	81
poly6	20.61	198	$3.6 \cdot 10^5$	-	1.193	198
poly8	25.98	114	$3.6 \cdot 10^5$	-	0.447	110
qspline	13.03	138	$6.8 \cdot 10^5$	-	0.517	133
sgfilter	5.51	56	$2.6 \cdot 10^5$	-	0.248	56

'-' means no feasible placement is found

In Table VII, we compare our fully pipelined approach to the time-multiplexer CGRA [21]. The fully pipelined architecture achieves $117.13\times$ average speedup.

TABLE VII
RESHAPE SC X TM [21]: PERFORMANCE IN GOPS/S.

	chebyshev	**mibench**	**poly5**	**poly6**	**poly8**	**qspline**	**sgfilter**
V2 [21]	0.9	0.8	1.3	1.3	1.4	1.7	1.5
our	173	143	121	53	156	122	198
speedup	192.2	178.7	93.1	40.8	111.4	71.8	132.0

IV. CONCLUSION

This work presents RESHAPE, a runtime data-flow hardware-based P&R for CGRA overlays using spatial parallelism. We can speed up by using replication to instantiate multiple copies of the P&R units and the data-flow graphs. We evaluate three overlay architectures and the trade-offs among mapping time and performance. We compare our P&R to VPR [11], CGRA-ME [27], and a time-multiplexer approach [20], [21]. RESHAPE outperforms previous works in the P&R execution time and CGRA performance. In future work, we will improve the mapping algorithm, the CGRA architecture using less routing resources, and provide a high throughput accelerator interface [22] for high-performance spatial mapping approaches.

ACKNOWLEDGMENTS

This work was supported by FAPEMIG, CNPq (Grant #436290/2018-9), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brazil (CAPES) – Financing Code 001, and Intel Academic Compute Environment.

References

[1] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, p. 118, 2019.

[2] L. B. D. Silva, R. Ferreira, M. Canesche, M. M. Menezes, M. D. Vieira, J. Penha, P. Jamieson, and J. A. M. Nacif, "Ready: A fine-grained multithreading overlay framework for modern cpu-fpga dataflow applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–20, 2019.

[3] N. Farahini, S. Li, M. A. Tajammul, M. A. Shami, G. Chen, A. Hemani, and W. Ye, "39.9 gops/watt multi-mode cgra accelerator for a multi-standard basestation," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 1448–1451.

[4] H. Lee, D. Nguyen, and J. Lee, "Optimizing stream program performance on cgra-based systems," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[5] M. Karunaratne, D. Wijerathne, T. Mitra, and L.-S. Peh, "4d-cgra: Introducing branch dimension to spatio-temporal application mapping on cgras," in *Int Conf on Computer-Aided Design (ICCAD)*, 2019.

[6] L. B. da Silva, D. Almeida, J. A. M. Nacif, I. Sánchez-Osorio, C. A. Hernández-Martínez, and R. Ferreira, "Exploring the dynamics of large-scale gene regulatory networks using hardware acceleration on a heterogeneous cpu-fpga platform," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–7.

[7] J. C. Penha, L. B. Silva, J. M. Silva, K. K. Coelho, H. P. Baranda, J. A. M. Nacif, and R. S. Ferreira, "Add: Accelerator design and deploy-a tool for fpga high-performance dataflow computing," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 18, p. e5096, 2019.

[8] R. Ferreira, W. Denver, M. Pereira, J. Quadros, L. Carro, and S. Wong, "A run-time modulo scheduling by using a binary translation mechanism," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. IEEE, 2014, pp. 75–82.

[9] T. Nowatzki, N. Ardalani, K. Sankaralingam, and J. Weng, "Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–15.

[10] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," in *FPL*, 2003.

[11] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, K. Kent, and J. Rose, "Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, Dec. 2011. [Online]. Available: https://doi.org/10.1145/2068716.2068718

[12] W. Carvalho, M. Canesche, L. Reis, F. Torres, P. Jamieson, L. Silva, J. Nacif, and R. Ferreira, "A design exploration of scalable mesh-based fully pipelined accelerators," in *International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020.

[13] C. Donovick, M. Mann, C. Barrett, and P. Hanrahan, "Agile smt-based mapping for cgras with restricted routing networks," in *Int Conf on ReConFigurable Computing and FPGAs (ReConFig)*, 2019.

[14] R. Ferreira, A. Garcia, T. Teixeira, and J. M. Cardoso, "A polynomial placement algorithm for data driven coarse-grained reconfigurable architectures," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2007, pp. 61–66.

[15] R. Ferreira, L. Rocha, A. Santos, J. Nacif, S. Wong, and L. Carro, "A run-time graph-based polynomial placement and routing algorithm for virtual fpgas," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013, pp. 1–8.

[16] M. V. Da Silva, R. Ferreira, A. Garcia, and J. M. Cardoso, "Mesh mapping exploration for coarse-grained reconfigurable array architectures," in *International Conference on Reconfigurable Computing and FPGA's (ReConFig)*. IEEE, 2006, pp. 1–10.

[17] G. Fontes, P. A. R. Silva, J. A. M. Nacif, O. P. V. Neto, and R. Ferreira, "Placement and routing by overlapping and merging qca gates," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.

[18] S. Yin, D. Liu, L. Sun, L. Liu, and S. Wei, "Dfgnet: Mapping dataflow graph onto cgra by a deep learning approach," in *ISCAS*, 2017.

[19] D. Liu, S. Yin, G. Luo, J. Shang, L. Liu, S. Wei, Y. Feng, and S. Zhou, "Data-flow graph mapping optimization for cgra with deep reinforcement learning," *IEEE Trans on CAD of Integrated Circuits and Systems*, 2018.

[20] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Resource-aware just-in-time opencl compiler for coarse-grained fpga overlays," in *Int Workshop on Overlay Architectures (OLAF) located in Int Symp on Field-Programmable Gate Arrays (FPGA)*, 2017.

[21] X. Li, A. K. Jain, D. L. Maskell, and S. A. Fahmy, "A time-multiplexed fpga overlay with linear interconnect," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[22] X. Li, K. Vipin, D. L. Maskell, S. A. Fahmy, and A. K. Jain, "High throughput accelerator interface framework for a linear time-multiplexed fpga overlay," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[23] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "Branch-aware loop mapping on cgras," in *Design Automation Conference (DAC)*, 2014.

[24] L. Josipović, S. Sheikhha, A. Guerrieri, P. Ienne, and J. Cortadella, "Buffer placement and sizing for high-performance dataflow circuits," in *Int Symp on Field-Programmable Gate Arrays (FPGA)*, 2020.

[25] M. Canesche, M. Menezes, W. Carvalho, F. Torres, P. Jamieson, J. A. Nacif, and R. Ferreira, "Traversal: A fast and adaptive graph-based placement and routing for cgras," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[26] R. Ferreira, J. M. Cardoso, A. Toledo, and H. C. Neto, "Data-driven regular reconfigurable arrays: design space exploration and mapping," in *International Workshop on Embedded Computer Systems*. Springer, 2005, pp. 41–50.

[27] M. J. P. Walker and J. H. Anderson, "Generic connectivity-based cgra mapping via integer linear programming," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 65–73.