A Design Exploration of Scalable Mesh-based Fully Pipelined Accelerators

Westerley Carvalho[1], Michael Canesche[1], Lucas Reis[1], Frank Torres[2], Lucas Silva[1], Peter Jamieson[3], José Nacif[1], Ricardo Ferreira[1]

[1]*: Computer Science Department, Universidade Federal de Vicosa, Brazil, email: ricardo@ufv.br*

[2]*: German Aerospace Center (DLR) , German, email:Frank.SillTorres@dlr.de*

[3]*: Dept. of Electrical and Computer Engineering, Miami University, Oxford, OH, USA, email: jamiespa@miamioh.edu*

***Abstract*—A dataflow graph is a computation abstraction with explicit dependencies that can be automatically parallelized. This work focuses on mapping dataflow graphs onto reconfigurable architectures and exploring them when fully pipelined. To embed these graphs onto mesh-based architectures, we propose a flexible mapping approach based on simulated annealing. We also implement a GPU parallel mapping to mitigate the mapping time. The trade-offs of target architectures areas are evaluated by exploring different interconnection topologies and local delay FIFOs on FPGAs and ASICs. To quickly evaluate different architectures, we developed a parameterized hardware generator that outputs costs in terms of wire length and buffer costs. We also propose a novel interconnection topology, called Chess. In comparison to other state-of-the-art mapping tools, including CGRA-ME, SAT solvers and VPR, our main contributions are: (a) finding optimal or near-optimal fully pipelined mappings; (b) scaling the dataflow graph size up to 70 operators without FIFOs; (c) proposing a framework to perform a design exploration of mesh architectures and more complex interconnection topologies.**

***Index Terms*—stream dataflow, placement, routing, reconfigurable, CGRA, fully pipelined architectures**

I. Introduction

Domain-specific hardware is a promising architecture choice to achieve power-efficient and high-performance computation. This work focuses on CGRAs as a one part of the heterogeneous computing landscape. In particular, recent work [1] present CGRAs processing spatial dataflows as an approach to exploit parallelism. In this work, we focus on improving the mapping algorithms and performing architecture exploration of the connecting topologies for mesh-based CGRAs.

The contribution of this work is twofold. First, we propose and evaluate a novel Simulated Annealing (SA) mapping algorithm for mesh-based CGRAs, including a new topology called *Chess*. We propose to search for a large SA solution space to find the best possible solution in reasonable amount of time. To explore and exploit CGRA architectures, we have created an experimental framework that takes in a parameterized configuration file and virtually creates, maps a dataflow to, and evaluates a CGRA architecture. We measure the quality of our results based on wire length and required FIFO stages to map a fully-pipelined design correctly. Our second significant contribution is to push the state-of-the-art for CGRAs, which can, currently, find optimal solutions for dataflow graphs up to 20 nodes, to up to 70 nodes. We compare our approach to three others: CGRA-ME [2], SAT solvers [3], and VPR [4]. We can also map graphs with up to 200 nodes by exploring interconnection architectures that provide flexibility to the mapping algorithm.

The remainder of this paper is structured as follows: Section II provides details on the CGRAs implementation and the respective algorithms that map designs to them. Section III describes the architecture topologies that we explore for CGRAs, and section IV describes our automated tool for creating an architecture and evaluating those architectures. Section V describes our mapping algorithm. Section VI shows results for this case study. Finally, section VII concludes the paper.

II. Background

A. Mesh, Wire Length, and Delay Mismatch

A mesh is a well-known low-cost and scalable interconnection network for CGRAs, where each cell has 4 adjacent local connections (except on the corners and borders). To calculate a cost to map a dataflow to a mesh, assume that local connections have wire cost=0, Figure 1(a-c) shows a dataflow design in (a) and various mapping instances in (b), (c), (d), noting that non-local routing wires are colored gray and local wires are bolded. In the (b) instance the longest wire (lw) has $cost = 1$, and total wire (tw) $cost = 2$. The tw includes both the mapped edges $a \rightarrow c$ and $c \rightarrow d$ (both having $cost = 1$). Figure 1(c) depicts an optimal mapping where $tw = 0$ since the mapping only uses local connections to neighbors. Figure 1(d) shows another example where the longest local wire $lw = 2$ and total wire $tw = 2$.

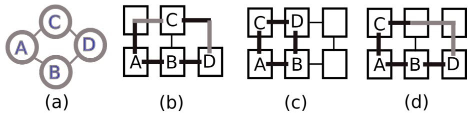


Fig. 1. (a) Dataflow; (b) lw=1,tw=2; (c) lw=tw=0; (d) lw=2,tw=2.

Most dataflow implementations take advantage of pipeline techniques to improve the maximum clock frequency for an architecture, and in a fully-pipelined dataflow mapping, there can be a delay mismatch between two or more paths if the signals have different arrival times. Figures 1(b) and 1(d) show two examples of a delay mismatch.

B. Delay FIFO

To matchup pipeline paths, we use delay FIFOs, as described in [5], which solves the delay mismatch. A delay FIFO with length L implements a variable size FIFO, where L is the maximum length. We fix the FIFO size at mapping time and implement the delay FIFO at the PE operation inputs. Figure 2(a) shows how to solve the mismatch with a FIFO $L = 2$ at node d for the example from Figure 1(d). It is possible to reduce the FIFO length to 1 by splitting the FIFO across the path $a \rightarrow b \rightarrow d$, as shown in Figure 2(b).

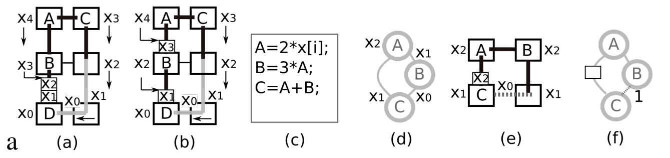


Fig. 2. (a) FIFO $L = 2$; (b) FIFO $L = 1$; (c) Stream x_i and x_{i+1}; (d) Dataflow pattern without optimal mapping in Mesh; (e) Minimal Mesh Mapping with FIFO; (f) Simplified mapped dataflow.

There are graph patterns for mesh-based architectures, where there is no optimal solution, i.e., without FIFOs to perform delay matching. Suppose we compute the convolution stream $c = 2 * x[i+1] + 6 * x[i]$, where the dataflow is shown in Figure 2(c-d). For this example, the minimal mapping requires at least a FIFO size 1, as shown in Figure 2(e). Finally, Figure 2(f) shows a simplified view of the dataflow. We add the label 1 to the long wire connection (edge $b \rightarrow c$) and use a block to represent a delay FIFO (edge $a \rightarrow c$).

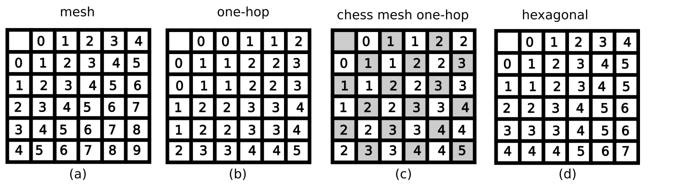
C. Mapping, Routing, Timing, and Metrics

In a stream-based system, the compiler performs Mapping, Routing, and Timing (MRT [5]). The mapping or placement phase (M) consists of assigning physical cells to the dataflow PE nodes. The routing phase (R) generates wiring paths to map the dataflow edges. Finally, the timing phase (T) guarantees that there are no delay mismatches. It is possible to perform these three phases separately or simultaneously. Nowatzki *et al.* [5] provides an analysis of these trade-offs for very limited graph sizes (≤ 20 nodes). In this work we evaluate dataflows ranging from 20 to 200 nodes, and show that even in the separation of phases approach, we can find optimal and near-optimal solutions with fast execution times in comparison to the results shown in [2], [3], [5].

For ASICs and FPGAs, a typical cost function metric is the total wire length as this impacts routing resource costs and hence the area of these chips. For CGRAs, the FIFO size is a more appropriate metric that measures the quality of mappings for a fully pipelined architecture [5]. The reasons for this are: first, the wire resources are predefined once an architecture is chosen, the routing capacity does not change, being set to bytes or larger (typically 16, 32, or 64-bit transfers of data). Regardless of the routing resources that are used in the CGRA, the final throughput will always be optimal for a valid solution, i.e., generating one result per clock cycle in a fully pipelined architecture. Second, adding FIFOs does not increase the latency because shorter paths are delayed to balance the longest path's overall delay. A routing solution that requires a few long wires will result in a timing solution that requires small FIFOs or none.

III. CGRA Interconnection Topologies

As already mentioned, there are simple dataflow graphs for which there is no possible mapping without FIFOs in a mesh topology. This section presents the one-hop, which is a well-know CGRA topology, and a novel topology named Chess. Previous work on CGRA design exploration [6] shows that one-hop mesh has enough flexibility to map dataflows onto CGRAs efficiently. In a one-hop mesh, each cell has 8 adjacent nodes.



mesh

	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9

(a)

one-hop

	0	0	1	1	2
0	1	1	2	2	3
0	1	1	2	2	3
1	2	2	3	3	4
1	2	2	3	3	4
2	3	3	4	4	5

(b)

chess mesh one-hop

	0	1	1	2	2
0	1	1	2	2	3
1	1	2	2	3	3
1	2	2	3	3	4
2	2	3	3	4	4
2	3	3	4	4	5

(c)

hexagonal

	0	1	2	3	4
0	1	2	3	4	5
1	1	2	3	4	5
2	2	3	4	5	6
3	3	3	4	5	6
4	4	4	5	6	7

(d)

Fig. 3. All distance from the top-left corner: (a) Mesh; (b) One-hop; (c) Chess; (d) Hexagonal.

One-hop mesh reduces the distances by a factor of 2 in comparison with mesh. Figures 3(a) and 3(b) show all wire distances (lw) from the top-left corner for mesh and one-hop, respectively. However, one-hop has double the number of local interconnections requiring significant hardware resources. Additionally, if we target optimal mapping without FIFOs, the advantage of one-hop is the increased number of adjacent nodes in comparison to mesh. One also might suggest using diagonal connections such as north-east or north-west. However, previous work has already shown that one-hop topologies produce the best results [6].

Our work proposes a novel hybrid topology named Chess (mesh/one-hop). Similar to a chessboard where there are black and white alternating cells, our Chess interconnection pattern has black cells with mesh connections, and white cells with one-hop connections. Although the chess pattern has, on average, 6 local connections per cell, the distance measure is quite similar to one-hop, as shown in Figure 3(c). Additionally we show a hexagonal or honeycomb pattern (see Figure 3(d)), which also has 6 adjacent cells. Moreover, the chess pattern reduces distance. For instance, the distance from the left top corner to the bottom right corner is 5 and 7 for the Chess and hexagonal, respectively (see Figure 3(c-d)). Furthermore, Chess performance is close to one-hop (8 connections) with the cost of a hexagonal (6 connections), as shown in our experiments.

IV. Code Generation for FPGA and ASICs

We also propose a tool to automatically generate the CGRAs, targeting the evaluation of area and power trade-offs against the two technologies - FPGAs and ASICs. Our tool is a parameterized hardware generator that evaluates the quality/cost of a dataflow mapping as a function of mesh size, interconnection topology, internal PE functionality, delay FIFO size, and the implementation cost of all of these on the target technology. Moreover, our CGRA design includes the configuration memories, and the resulting design is partially reconfigurable at run-time.

Each CGRA consists of a heterogeneous set of PEs and an interconnection network that defines which PEs connect to it. Each PE has a functional unit (FU) and a wrapper interconnection interface. For each PE, the configuration file specifies: 1) word size; 2) delay FIFO length, 3) functional unit operation and opcodes; 4) neighborhood interconnection with other PEs.

We evaluate the topologies and delay FIFO impact on FPGA and ASICs to implement different CGRAs. Figure 4 shows the area for CGRAs with different configurations implemented as an overlay in FPGA technology. We use Intel's Quartus for the synthesis and target an ARIA10 FPGA.

The area results are then normalized to a baseline as follows. For each grid size, our baseline is the mesh topology without any FIFOs and bypassing routing resources. We considered grid sizes ranging from 81 up to 1,296 PEs.

The Chess and one-hop wires and muxes add an overhead of 1.2 to 1.3× and 1.4 to 1.5×, respectively. Therefore, these topologies that potentially do not need FIFOs can improve the area overhead. Finally, the clock frequency for all designs ranges from 250 MHz to 450 Mhz. All of these designs have a 16-bit word size. Furthermore, we design a 4-bit CGRA with a heterogeneous set of PEs to use all the FPGA DSP units. Half of the PEs have 8 operations that include multipliers, and the other half have no multipliers. The 46×64 CGRA use 40% of the FPGA ALMs with a total of 3,036 PEs, and this CGRA achieves a peak performance of 1.2 Tera 4-bits ops.

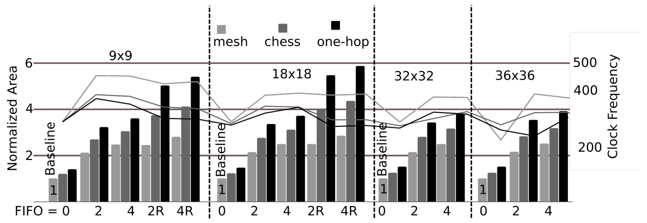


Fig. 4. Normalized area in number of ALMs for FPGA Overlay design.

We synthesize a set of CGRA designs by using the ASIC flow FreePDK45TM 45nm variant of the FreePDKTM process design kit [7]. Figure 5 shows the total silicon area for the architectures and the fraction of the area occupied by the registers in black. First, we use the 18×18 design without routing resources and delay FIFOs as a baseline. The 18×18 is 3.8× bigger than the 9×9, which shows that the design grows linearly from 81 to 324 PEs. The Chess and one-hop wires and muxes add a small overhead of 1.05× and 1.08×, respectively. Therefore, the topologies without FIFOs improve the mapping without an area overhead. The clock frequency is approximately 1 GHz for these architectures.

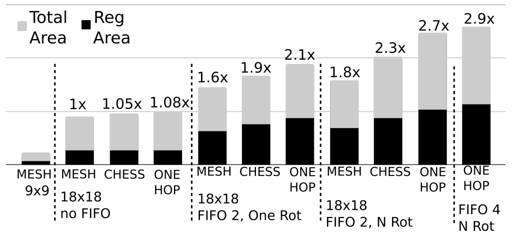


Fig. 5. Total area and register area for CGRA (ASIC Flow).

V. Mapping Dataflows with Simulated Annealing

Recent works [2], [5], [8] propose integer linear programming (ILP), SAT solvers, and deep reinforcement learning for CGRA mapping. However, even when using state-of-the-art ILP and SAT solvers, the execution time is in the order of minutes. We propose to use a simulated annealing (SA) approach, which is a well-known approach for mapping designs to CGRAs [9] and FPGAs [4]. Instead of performing random swaps, we propose a heuristic to scan the grid and perform the swaps sequentially.

Our approach performs only the SA mapping with local routing. Next, we execute the detailed routing for the long wires, and finally, we perform the timing step where we consider the FIFO cost function. We also propose to explore the solution space by performing multiple executions of the full SA. Each instance starts from a random permutation of an initial mapping, and the SA minimizes the wire length for each instance. As mentioned in Section II-C, the wire length is just a mapping guideline, and our goal is to minimize the FIFO size. The mapping does not consider the delay matching requirements, and therefore, the reduced complexity needs less execution time. Furthermore, we implement a GPU version of our SA, where we execute each SA in its thread, differently from previous approaches that parallelize the internals of an SA instance with multiple threads.

VI. Experimental Results

We evaluated our mapping technique by using public domain benchmarks extracted from three related works: (a) the CGRA-ME design exploration framework [2]; (b) a Mediabench set from the University of California, Santa Barbara [10];(c) Stream-Join dataflow control model [11]. We include the Stream-Join for irregular flows [11], which handles sparse data with simple dataflow operators and fewer dataflow edges and nodes. We evaluate four dataflow examples from this set [11]. In total, we evaluate 37 benchmarks: 14 from CGRA-ME [2], 8 stream-joint graphs [11], and 15 from Mediabench [10]. In all the experiments we map to a square array of PEs where an $N \times N$ array has an $N = \left\lceil \sqrt{|Nodes|} \right\rceil$.

First, we propose to compare our approach to the ILP-based approach from [2] and the SAT-solver approach presented

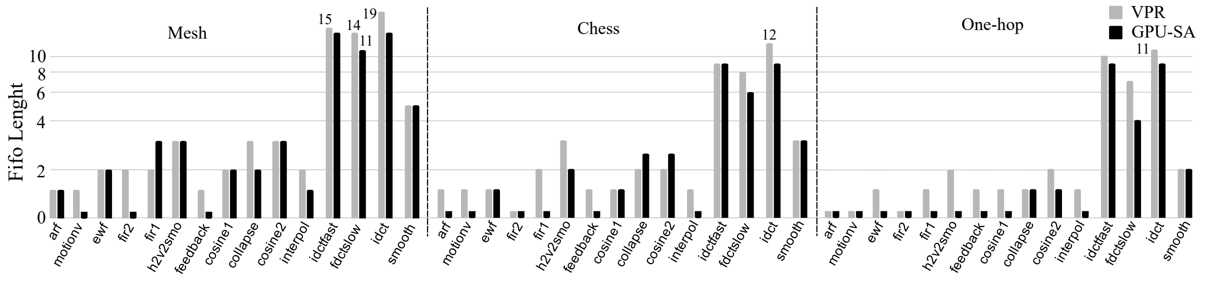


Fig. 6. FIFO length for UCSB express suite dataflows [10] in three topologies for 1,000 Instances of VPR and our SA approach. Low is better.

in [3]. With a timeout of 10 minutes, the ILP approach [2] maps only 22 of the 37 benchmarks with the maximum graph size of 23 nodes, and the SAT approach [3] maps 25 with the maximum graph size of 44. Our approach successfully maps all 37 benchmarks with the maximum graph size of 196, and the average mapping time is a few seconds.

CGRA-ME [12] is a framework created for design exploration for CGRA architectures, similar to ours. Although their mapping approaches are generic, the execution time for the current mapping based on ILP and SA [2] are prohibitively long (minutes or hours) for dataflows with more than 20 nodes.

As ILP/SAT solvers do not scale, we propose to compare our SA approach to the VPR tool [4]. Like our approach, VPR does not perform the timing step for fully pipelined CGRAs. Therefore, we apply our routing and our timing approach to the VPR placement. We execute 1,000 VPR instances with random initial placements to perform a fair comparison. Finally, as the two benchmark sets from [2], [11] are limited to 25 nodes, we use the UCSB mediabench set [10], in addition to *k-means* benchmarks to compare the SA approaches. This suite is a representative graph selection from *Mediabench*, where the graph size ranges from 10 to 300 nodes.

Figure 6 shows the best mapping in 1,000 instances for 15 dataflows in three topologies: mesh, the novel Chess, and one-hop. Our mapping on one-hop reaches the optimal solution for all graph sizes smaller than 66 nodes, while VPR [4] reaches the optimal solution for graph sizes smaller than 32 nodes. The average FIFO length for the proposed Chess topology is 2.2 compared to 1.7 obtained by the one-hop and 4.1 for the mesh. Therefore, the Chess reduces the one-hop cost as shown in Section IV, and reaches a closer performance to one-hop and 1.9x better than mesh. Between 72 and 102 nodes, there are 3 graphs where one-hop/Chess only require FIFO length of one. For Mediabench [10], in comparison to VPR, our mapping reduces the average FIFO length in 1.22x, 1.42x, and 1.58x for mesh, Chess, and one-hop topologies, respectively.

VII. Conclusion and Future Work

In this work, we performed a design exploration of CGRAs using a SA algorithm and a novel Chess topology. Our results show that our algorithm is flexible and efficient to explore the solution design space for larger dataflows compared to those previously studied. The fully pipelined mapping has the challenge of the path delay mismatch. Therefore, even for small graphs, the problem is hard to solve. We can map dataflows with 20 to 70 nodes in larger CGRAs (≥ 16) without delay FIFOs due of the extra connections in the Chess and one-hop topologies. Previous works [5] argues that it is hard to map graphs with FIFO of sizes 2, 3 or less and our results show that our SA approach performs it successfully. Finally, our mapping is straightforward to parallelize on a GPU , and it requires a few seconds to perform mappings for 200 nodes.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brasil (CAPES)– Finance Code 001, FAPEMIG, CNPq, and Intel.

References

[1] 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 CAD of Integrated Circuits and Systems*, 2020.

[2] M. J. Walker and J. H. Anderson, “Generic connectivity-based cgra mapping via integer linear programming,” in *Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[3] 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)*. IEEE, 2019.

[4] 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 TRETS*, vol. 4, no. 4, p. 32, 2011.

[5] T. Nowatzki, N. Ardalani, K. Sankaralingam, and J. Weng, “Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign,” in *ACM PACT*, 2018.

[6] N. Bansal, S. Gupta, N. Dutt, A. Nicolau, and R. Gupta, “Network topology exploration of mesh-based coarse-grain reconfigurable architectures,” in *DATE*, vol. 1. IEEE, 2004.

[7] FreePDK45, “Freepdk45tm 45nm process design kit,” https://www.eda.ncsu.edu/wiki/FreePDK45:Contents, 2019.

[8] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[9] 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 *International Conference on Field Programmable Logic and Applications*. Springer, 2003, pp. 61–70.

[10] S. B. University of California, “Express benchmarks,” https://web.ece.ucsb.edu/EXPRESS/benchmark/, 2020.

[11] V. Dadu, J. Weng, S. Liu, and T. Nowatzki, “Towards general purpose acceleration by exploiting common data-dependence forms,” in *International Symposium on Microarchitecture*, 2019.

[12] U. of Toronto, “Cgra - modelling and exploration,” http://cgra-me.ece.utoronto.ca/, 2019.