Efficient Application Mapping on CGRAs based on Backward Simultaneous Scheduling/Binding and Dynamic Graph Transformations

Thomas Peyret, Gwenole Corre, Mathieu Thevenin, Kevin Martin, Philippe Coussy

Efficient Application Mapping on CGRAs based on Backward Simultaneous Scheduling/Binding and Dynamic Graph Transformations

Thomas Peyret, Gwenolé Corre, Mathieu Thevenin
CEA, LIST, Laboratoire Capteurs et Architectures Électronique
F-91191 Gif-sur-Yvette, France
firstname.lastname@cea.fr

Kevin Martin, Philippe Coussy
Université de Bretagne-Sud, Lab-STICC
Lorient, France
firstname.lastname@univ-ubs.fr

***Abstract*—Mapping an application on a coarse grained reconfigurable architecture (CGRA) is a complex task which is still often completely or partially realized manually. This paper presents an automated synthesis flow based on simultaneous scheduling and binding steps. The proposed method uses a backward traversal of the formal model obtained after compilation and dynamically transforms it when needed. Our approach is compared with state of the art techniques and its interest is shown through the mapping of several applications from digital signal and image processing domain.**

Keywords—CGRA; Mapping; Scheduling; Binding

I. INTRODUCTION

For the last two decades, Coarse Grained Reconfigurable Architectures (CGRAs) have been mainly proposed for accelerating multimedia applications. CGRA is usually composed of Processing Elements (PE), named tiles, which communicate through an interconnection network. CGRA are interesting trade-offs between FPGAs and many-core architectures [1]. Many works [2, 3, 4, 5, 6] proposed hardware architectures that distinguish by different features like homogeneous/heterogeneous tiles, absence/presence of Register Files (RF), RF sizes, kind of operators (×, +, -…) or interconnection networks (mesh, torus…). For example, Fig. 1 represents a CGRA composed of homogeneous tiles including register files, interconnected through a 2D torus mesh network.

Executing an application on a CGRA requires its operations to be scheduled and bound on tiles and its data to be assigned to registers. Mapping process, that has to respect control and data dependencies of the application, is a complex task that cannot be done by hand anymore: it has to be automated. For that purpose, application is generally formally described through a Control Data Flow Graph (CDFG) after compilation and CGRA is abstracted through a formal model. The target architecture strongly constrains the mapping process which objective is to maximize timing performances, by minimizing latency and/or maximizing throughput. Scheduling and binding are known to be NP-complete [7]. Many approaches that can be classified into four main categories have been proposed to tackle these problems. They differ on the way they treat scheduling and binding steps (sequentially or concurrently) and methods they use (exact or approximate). The first category proposes to solve separately scheduling and binding by using heuristics or meta-heuristics as in [7, 8, 9]. The second one also proposes to solve scheduling and binding sequentially but by combining heuristics and exact methods as in [10, 11] ([11] enables to cope with RFs unlike many other methods). These two works rely on static transformations of the Data Flow Graph (DFG) to help mapping applications. These transformations allow for a better exploration of the solution space but are done *a priori*, *i.e.* before binding, thanks to a cost function. The third category tries to solve the whole problem by using exact methods [12, 13]. Unfortunately, these methods typically do not scale up as shown in [7]. The fourth one targets the complete problem by leveraging on meta-heuristics like in [14, 15] ([15] supports RFs) where a simulated annealing procedure is used. However, simulated annealing usually converges very slowly toward potential optimal solution.

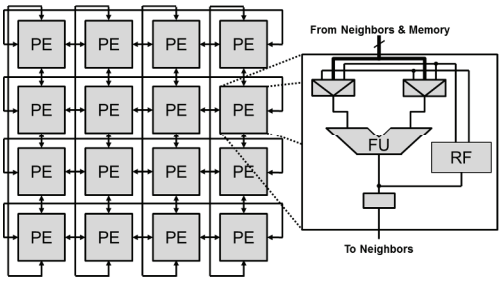

Fig. 1. A 4 × 4 CGRA with a 2-D torus mesh and a RF in each tile.

In this paper we present a unified approach that combines a heuristic and an exact method. It allows for mapping applications to various types of CGRAs by using a backward simultaneous scheduling and binding combined with dynamic graph transformations. A list-based scheduling algorithm is coupled with a binding procedure based on a simplified Levi's algorithm [16]. Solution space is further efficiently explored by traversing the graph backward and by transforming it dynamically when needed which early prevents searching for dead-end solutions.

The rest of the paper is organized as follows. Section II introduces our method. Section III presents the experiments and discusses the results. Conclusion is given in section IV.

II. PROPOSED METHOD

The proposed design flow is presented in Fig. 2. Input is a purely functional specification written in C/C++ and the targeted CGRA model. First, the application is compiled to obtain a Control and Data Flow Graph (CDFG). Then, CDFG and CGRA models are used to generate mappings. Proposed algorithm allows the exploration of the solution space thanks to the combination of graph transformations, a heuristic-based scheduling and a binding leveraging on an exact though simplified method with a pruning step. CDFG is dynamically transformed to find a solution during scheduling and binding. As done in [10], two types of transformations are proposed: routing to keep data dependencies and splitting to reduce interconnection pressure. Our algorithm "reroutes" a node when scheduling fails and choose between routing and splitting when binding fails (see section II.B.3). The purpose of the synthesis flow is to optimize latency under resources constraint.

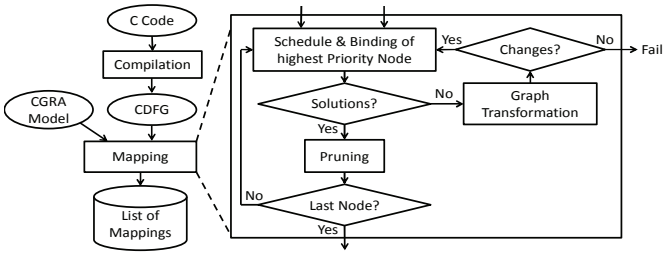

Fig. 2. General flow and Algorithm core.

A. Application and Architecture Modeling

CDFG is composed of a Control Flow Graph (CFG) and a set of basic blocks represented by DFGs. A DFG is a bipartite directed acyclic graph composed of data, represented by a rectangle in Fig. 3(c) and Fig. 4, and operation nodes (circles in Fig. 3(c) and Fig. 4) and arcs. In addition to computation node (×, +, -…), another operation node is defined: memorization. A memorization node makes explicit data dependencies along cycles – *e.g.* in the DFG in Fig. 3(c), node 2' is a memorization node that makes explicit the data dependency between nodes 2 and 4 over one clock cycle.

CGRA is modeled by a bipartite directed graph with two types of nodes: operators and registers. Timing is implicitly represented by connections between registers and operators. In this model, two subtypes of operator nodes are defined: conventional and memorization operators. Conventional operator represents any physical implementation of operation (×, +, -…). Memorization operator is associated to a register and makes explicit the storage of a value. Connection between output register nodes and conventional operators depends on the interconnect network. An example of this representation is given in Fig. 3(a) and (b). This CGRA graph model is very versatile and can represent CGRA with:

- homogeneous or heterogeneous tiles;
- presence or absence of register file;
- shared or local register file;
- homogeneous or heterogeneous operators;
- regular or specific interconnect network;
- operators that require one cycle or more.

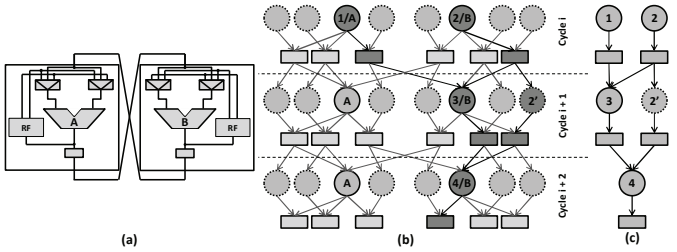

Fig. 3. **(a)** a 2 × 1 CGRA with 2 registers in RF, **(b)** Equivalent graph model on three cycles, **(c)** DFG model. In **(b)**, a possible mapping of DFG in **(c)** is represented in dark grey. Memorizations are dotted and registers rectangular.

To use methods from graph theory domain, the following equivalences between nodes from DFG and CGRA graph are defined: computation node ⇔ conventional operator, memorization operation ⇔ memorization operator and data node ⇔ register node. Hence, these models are homomorphic and binding a CDFG on a CGRA comes down to find each DFG in the CGRA graph [10]. A mapping illustration is given in Fig. 3(b).

B. Mapping Algorithm

Proposed approach merges scheduling and binding to avoid the drawbacks of sequential approaches like [7, 8, 9, 10, 11]. The main idea is to schedule an operation node and immediately check if there is at least one binding solution. In this case, the next operation node is scheduled, else the graph is transformed. A pruning pass is executed at the end of each scheduling cycle to reduce the number of partial bindings.

1) Scheduling: Proposed approach traverses backward the graph and uses a list scheduling algorithm in which the schedulable operations are listed by priority order. The priority of a node is inversely proportional to its timing mobility. Mobility is defined by applying ASAP and ALAP scheduling as described in [17]. However, for nodes with an identical mobility, the number of successors is considered (more successors, higher priority). Indeed, a node with many successors is more difficult to map and scheduling it first allows for optimizing the final latency (*e.g.* in Fig. 4(c), node 2 has a higher priority than node 1). Considering backward traversal, a node is schedulable if all its successors are already scheduled (*e.g.* node 2 in Fig. 4(b) is not schedulable because node 3 is not yet scheduled).

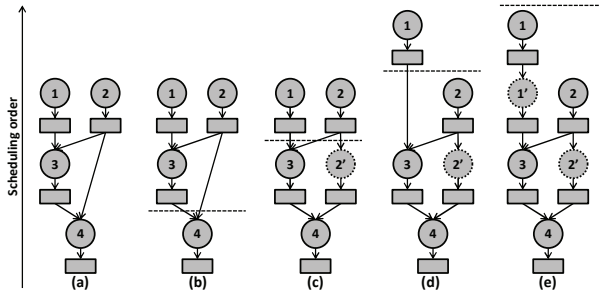

Fig. 4. Illustrative example of scheduled and transformed DFG on a one tile CGRA. Horizontal line shows the limit between scheduled and non scheduled nodes. Memorization nodes are dotted circles. **(a)** Initial DFG, **(b)** After scheduling node 4, **(c)** After scheduling node 3 and routing node 2, **(d)** After scheduling node 2, **(e)** Scheduled DFG after routing and scheduling node 1.

2) Binding: Unlike approaches as [10] which use a regular version of Levi's algorithm, we modify Levi's algorithm by making it incremental. Hence, it uses the partial bindings previously found (and kept by the pruning step) to search for every possible new binding including the currently scheduled node (*e.g.* from the partial bindings obtained in Fig. 4(c), the algorithm finds every possible binding with node 2). The resulting partial solutions are used in turn for the next node.

3) Graph transformations: Transformations occur when a node is not schedulable (*e.g.*, in Fig. 4(c), node 2 is not schedulable and thus memorization node 2' has to be added) or when the binding algorithm finds no solution for the current node. In the proposed approach, as opposed to works like [10] which realizes *a priori* tranformations, the graph is transformed *dynamically* when required. The three following graph transformations are available, as illustrated in Fig. 5:

a) Simple route: used if the node is not schedulable or in absence of available operating ressources. A memorization node is added to delay node scheduling as in Fig. 5(c).

b) Node splitting: if an operation node has no binding solution because the number of output edges is too high (*e.g.* non reachable successors or number of output edges greater than the architecture possibilities), two transformations are available to reduce this number by adding another node with the same predecessors to the current cycle and distributing outputs edges as fairly as possible between the two nodes:

- operation node splitting, equivalent of "recomputing" in [10] and illustrated in Fig. 5(b);
- memorization node splitting, illustrated in Fig. 5(d).

Node splitting is privileged if there are enough free resources and if the node has more than one successor. Otherwise, simple route is used. In case of forward scheduling, the only relevant transformation would be simple route. Indeed, in a forward scheduling, the identification of the successor nodes that would be really scheduled at next cycle is not possible. Thus, edge distribution during node splitting would be purely arbitrary and would not provide a real gain over simple route.

4) Pruning: The prunning step is introduced because Levi's algorithm is exhaustive and the partial mappings number can thus be very high depending on data dependencies and CGRA constraints. It removes redundant partial mappings as soon as no more node can be scheduled in the current clock cycle. A partial mapping is redundant if it uses exactly the same operators to make the same operations than another partial mapping at the current cycle.

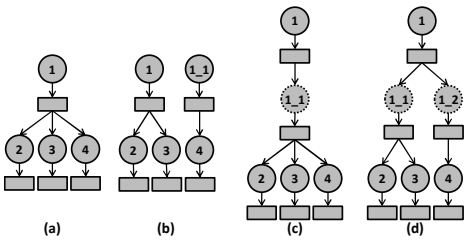

Fig. 5. **(a)** Initial DFG, **(b)** operation node splitting on node 1 from **(a)**, **(c)** simple route on node 1 from **(a)**, **(d)** memorization node splitting on node 1_1 from **(c)**.

III. EXPERIMENTS AND RESULTS

A. Experimental setup

The method presented in this paper has been fully automated and implemented using Java and Eclipse Modeling Framework. Compilation step uses GCC-4.7.2 to generate the CDFGs of applications that are used as input. Nine algorithms from signal processing applications and High Level Synthesis (HLS) benchmarks were used for experiments. They are: Discrete Cosine Transform (DCT-2D), matrix product, Fast Fourier Transform (FFT), Manhattan Distance, Exponential Moving Average Filter (EMA), Moving Window Deconvolution (MWD), unsharp mask, elliptic filter and a low-pass filter (DC Filter). A workstation integrating an Intel Xeon and 8 GB of RAM was used to run the program.

The proposed approach is compared to two approaches from the state-of-the-art. *"Method 1"* is a method that solves scheduling and binding separately as in the initial step of [7]. It uses a forward list scheduling algorithm. For fair comparisons, binding is made by using Levi's algorithm. *"Method 2"* applies *a priori* graph transformations and tries to find a mapping with Levi's algorithm as proposed in [10, 11]. Since [10] and [11] have been shown to provide better results than [14] and [15], we do not compare with these approaches in this paper.

To obtain a large spectrum of results, several parameters have been varied: CGRA size, RF size and the number of tiles the final mapping is allowed to use – *i.e.* 16 sets of constraints per C code per method. Four metrics are considered to assess the quality of the different methods:

- *Success rate*: defined as the percentage of time that a method finds a solution when at least one of the three methods succeeds.
- *Latency:* the best latency provided by the methods.
- *Diversity:* defined as the number of different mappings that has been found *i.e.* the ability of a method to widely explore the solution space. Two mappings are said different when they use different tiles or the interconnection network in a different way.
- *Efficiency:* defined by the ratio between the number of different mappings and the computation time.

B. Results

Fig. 6 to Fig. 9 present results for each considered application. As shown in Fig. 6, Method 1, which solves scheduling totally independently from binding, leads to the lower success rate (~37%). Method 2, which applies *a priori* transformations, gives better results (~62%). Our approach gives the best success rate (~99%). Since the proposed method relies on a heuristic-based scheduling algorithm, the best latency cannot always been found. However, as shown in Fig. 7 it provides the best latencies more than twice as much as the other methods (~90%). When the best latency is not found, it is increased by 1.5 cycles in average, *i.e.* a mean increase of 15% of these latencies. Fig. 8 shows that proposed approach allows finding more different mappings than Method 1 and Method 2 (respectively 3.7 and 2.4 times more) and thus having higher exploration quality. Fig. 9 illustrates that exploration is more efficient because the time spend by mapping is lower than for the other methods – respectively 2.6 and 2.2 times more.

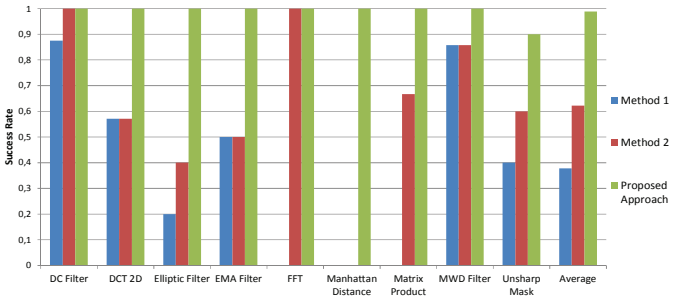

Fig. 6. Success rate.

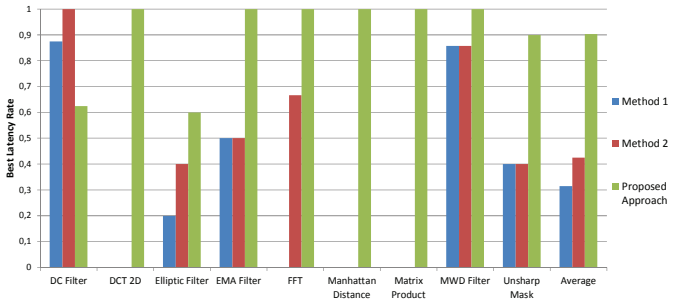

Fig. 7. Best latency rate.

All these experiments show the interest of the proposed approach which provides better results in terms of quality, diversity and efficiency.

IV. Conclusion

A method to map C code on a generic CGRA architecture is presented in this paper. This approach, which relies on a GCC front-end and a CGRA model, explores the solution space by solving simultaneously the scheduling and binding problem through a backward traversal of the application graph, allowing transforming the application graph only when needed. Experimental results shown that our method has the highest success rate, finds most of the time the best latencies and better explores the solution space than the state of the art methods.

References

[1] M. B. Taylor, "Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse," in *Proc. DAC*, 2012.

[2] R. David, D. Chillet, S. Pillement, and O. Sentieys, "DART: a dynamically reconfigurable architecture dealing with future mobile telecommunications constraints," in *Proc. Parallel and Distributed Processing Symposium*, 2002.

[3] M. Lanuzza, S. Perri, P. Corsonello, and M. Margala, "A new reconfigurable coarse-grain architecture for multimedia applications," in *Proc. Adaptive Hardware and Systems, Second NASA/ESA Conference on*, 2007.

[4] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *Computers, IEEE Transactions on*, vol. 49, no. 5, pp. 465–481, 2000.

[5] F. Campi, A. Deledda, C. Mucci, A. Lodi, M. Pizzotti, L. Cirrarelli, P. Rolandi, A. Vitkovski, and L. Vanzolini, "A dynamically adaptive DSP for heterogeneous reconfigurable platforms," in *Proc. DATE*, 2007

[6] S. Shukla, N. W. Bergmann, and J. Becker, "QUKU: A fast run time reconfigurable platform for image edge detection," *Reconfigurable Computing: Architectures and Applications*, vol. 3985, pp. 93–98, 2006.

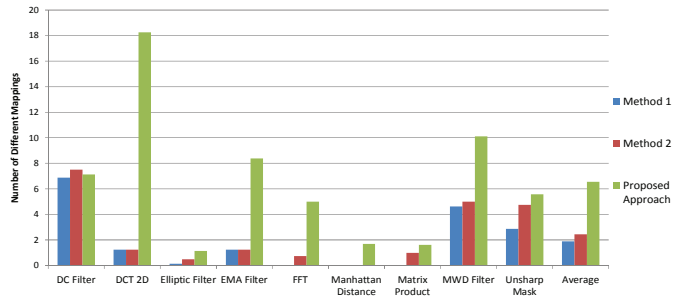

Fig. 8. Average number of different mappings.

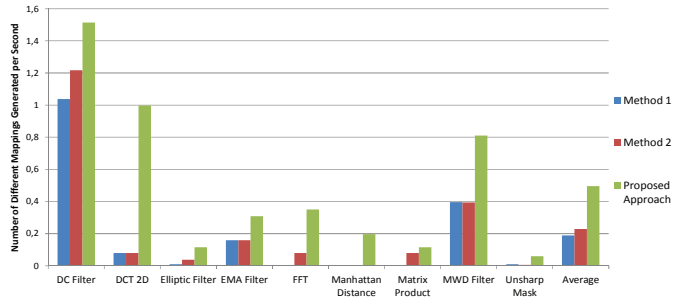

Fig. 9. Average number of different mappings generated per second.

[7] G. Lee, K. Choi, and N. D. Dutt, "Mapping multi-domain applications onto coarse-grained reconfigurable architectures," *Computer-Aided Design of Intergrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 5, pp. 637–650, 2011.

[8] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-S. Kim, "Edge-centric modulo scheduling for coarse-grained reconfigurable architectures," in *Proc. Parallel architectures and compilation techniques, 17th international conference on,* 2008.

[9] S. Friedman, A. Carroll, B. Van Essen, C. Ebeling, S. Hauck, and B. Ylvisaker, "SPR: an architecture-adaptive CGRA mapping tool," in *Proc. Field programmable gate arrays, ACM/SIGDA international symposium on,* 2009.

[10] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: using epimorphism to map applications on CGRAs," in *Proc. DAC*, 2012.

[11] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "REGIMap: register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *Proc. DAC*, 2013.

[12] J. A. Brenner, J. C. van der Veen, S. P. Fekete, J. Oliveira Filho, and W. Rosenstiel, "Optimal Simultaneous Scheduling, Binding and Routing for Processor-like Reconfigurable Architectures," in *Proc. Field Programmable Logic and Applications, International Conference on*, 2006.

[13] E. Raffin, C. Wolinski, F. Charot, K. Kuchcinski, S. Guyetant, S. Chevobbe, and E. Casseau, "Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture," in *Proc. Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2010.

[14] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "DRESC: A retargetable compiler for coarse-grained reconfigurable architectures," in *Proc. Field-Programmable Technology, IEEE International Conference on*, 2002.

[15] B. De Sutter, P. Coene, T. Vander Aa, and B. Mei, "Placement-and-routing-based register allocation for coarse-grained reconfigurable arrays," *ACM SIGPLAN Notices*, vol. 43, no. 7, p. 151, Jun. 2008.

[16] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcolo*, vol. 9, no. 4, pp. 341–352, Dec. 1973.

[17] D. D. Gajski and L. Ramachandran, "Introduction to high-level synthesis," *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 44–54, Jan. 1994.