A Scalable Design Approach to Efficiently Map Applications on CGRAs

Satyajit Das, Kevin Martin, Philippe Coussy, Thomas Peyret, Gwenolé Corre, Mathieu Thevenin

A Scalable Design Approach to Efficiently Map Applications on CGRAs

Satyajit Das, Kevin Martin, Philippe Coussy
Univ. Bretagne-Sud, CNRS UMR 6285, Lab-STICC
F-56100 Lorient, France
firstname.lastname@univ-ubs.fr

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

***Abstract*— Coarse-Grained Reconfigurable Architectures (CGRAs) are promising high-performance and power-efficient platforms. However, their uses are still limited because of the current capability of the mapping tools. This paper presents a new scalable efficient design flow to map applications written in high level language on CGRAs. This approach leverages on simultaneous scheduling and binding steps respectively based on a heuristic and an exact method stochastically degenerated. The formal graph model of the application, obtained after compilation, is backward traversed and dynamically transformed when needed to allow for a better exploration of the design space. Results show that our approach is scalable, finds most of the time the best solutions i.e. the mappings with the shortest latencies, achieves lowest failure rate in carrying out solutions, provides lower computation time and explores more efficiently the solution space than the state of the art methods.**

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. CGRAs are indeed an interesting trade-off between FPGAs and many-core architectures because their power efficiency is close to nonprogrammable hardware accelerators while they are programmable [1]. The literature is very rich in CGRAs architectures [2–6], which distinguish by different features such as the granularity of the Processing Elements (PE), homogeneity or heterogeneity of PE, type of operators (ALU, multiplier, adder, shifter…), absence/presence/size of Register Files (RF) or interconnection network topologies (mesh 2-D, tore…). PE, also named *tile*, is usually composed of a functional unit, a local RF, two input multiplexers and an output register as illustrated in Fig. 1. Interconnection network is traditionally a 2D mesh that offers the possibility to communicate with four neighbors. However, additional column or line buses, or more complex networks as segmented buses can be used to increase the communication capability of each tile [2], [4], [5].

The result of the compilation of an application on a CGRA (also named *mapping*), is the scheduling and the binding of its operations on operators and registers. This process, that has to respect control and data dependencies of the application and the architecture constraints of the CGRA, is a complex task still often realized by hand. However, applications are increasingly more complex and dealing with hundreds or thousands of operations is simply not reasonable: the mapping process must be automated. Scheduling and binding are known to be NP-complete [7] and obtaining a good solution requires to explore deeply the solution space. Indeed, the target architecture strongly constrains the mapping process which objective is to maximize timing performances (minimize latency and/or maximize throughput).

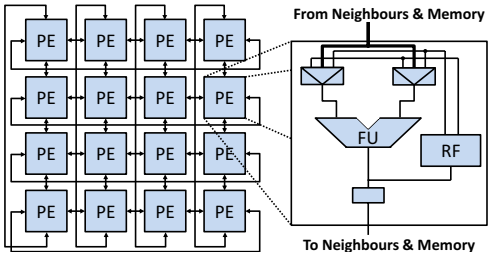

Fig. 1. A 4×4 CGRA with torus mesh an FU and RF in each tile.

In this paper, a scalable approach that allows for mapping entire applications onto various types of CGRAs is presented. The solution space is efficiently explored by traversing backward the formal intermediate representation (IR) of the application, by combining scheduling and binding steps and by transforming IR dynamically when needed.

The rest of the paper is organized as follows. Section II presents the related works. Section III depicts proposed method. Section IV presents the experiments and discusses obtained results. Conclusion is given in Section V.

II. RELATED WORK

As CGRAs are classically used to speed up kernel loops, most of the literature methods tries to minimize the initiation interval of loops kernel [7–11] *i.e.* maximize throughput. Existing techniques can also be used to compile entire application code onto CGRA. Mapping methods can be roughly divided into two main categories which differ on the way scheduling and binding steps are realized, i.e. sequentially or concurrently.

The first category simplifies the mapping problem by solving scheduling and binding sequentially with heuristics and/or meta-heuristics [7–9] or exact methods [12], [13]. In [8] and [9], iterative modulo scheduling heuristic [14] is used for scheduling. In [8] an edge-based binding heuristic is used (instead of classical node-based approaches) to reduce the

number of fails. In [9], binding problem is tackled by combining a routing heuristic from FPGA synthesis and a Simulated Annealing (SA) algorithm for placement. Schedulable operation nodes are moved randomly according to a decreasing temperature and a cost function. The main limitation of this kind of approaches is the compromise that has to be done in order to have a reasonable computation time. Indeed, to efficiently explore the solution space, the temperature requires to be high and to decrease slowly, leading to very slow convergence time. Conversely, if the temperature is low or decreases rapidly, the solution is hardly ever optimal and corresponds nearly always to a local optimum. The method presented in [7], which provides better results, starts by finding a solution on a simplified problem with heuristic-based methods for both scheduling and binding and then tries to improve the initial solution with a genetic approach. The initial solution is found by using a list-scheduling algorithm and the binding is realized by considering only a single column of the CGRA. Then the genetic algorithm optimizes the first solution by moving operations to minimize the latency. However, this method uses only one seed which limits its ability to explore the whole solution space. Moreover, as every guided stochastic algorithm, this technique suffers from both low run time efficiency and low convergence rate to high quality results. In [12], [13], authors solve the scheduling and the binding problems sequentially by using respectively a heuristic and an exact method. Scheduling is made implicitly by integrating both architectural constraints (i.e. the number of operations simultaneously executable on the CGRA and the maximum out-degree of each operation due to the communication network) and timing aspect into the DFG by statically transforming it. Two transformations are proposed: "recomputing" that duplicates computation nodes and "routing" that duplicates data nodes to make explicit the conservation of a result. These transformations hopefully facilitate the application's mapping. Binding is done by finding the common sub-graph between the transformed DFGs and a time extended CGRA with Levi's algorithm [15]. However, since the graph transformations are done *a priori*, it is very difficult to know which transformation is relevant at a given time. This reduces the ability of the method to efficiently explore the solution space since the problem is over-constrained. In [19], an algorithm based on graph transformation also is used to find the homomorphism between the graphs.

The second category solves the scheduling and binding problems as a whole. Hence, [7], [16], [17] use exact methods, *e.g.* ILP-based algorithms, to find optimal results. Unfortunately, these methods suffer from scalability issues as illustrated in [7]. The method presented in [10], and its extension that can cope with RFs [11], leverage on meta-heuristics. They are based on a Simulated Annealing (SA) framework that is also a guided stochastic algorithm. The classical placement and routing problem which can be solved with SA, is there extended in three dimensions to include scheduling. Thus schedulable operation nodes are moved absolutely randomly through time and space. The convergence is there even slower than for other SA method because it includes scheduling.

The key idea of the proposed mapping approach is to combine the advantages of exact, heuristic and meta-heuristic methods while offsetting as much as possible their respective drawbacks. Hence, as detailed in the next section, scheduling and binding problems are solved simultaneously by using a heuristic-based algorithm and a randomly degenerated exact method respectively and by transforming the formal model of the application dynamically.

III. PROPOSED METHOD

The proposed design flow is presented in Fig. 2. Inputs are a C/C++ application code and the targeted CGRA model. The objective of the method is to minimize the latency of the DFG under resource constraint by deeply exploring the design space while keeping a low mapping delay ("synthesis time"). The output of the flow is the mapping with the best latency that has been found during the exploration of the design space *i.e.* one of the mappings with the best latency since our flow is inherently able to find several mappings with the best latency. The description of the application is first compiled to obtain a formal Control and Data Flow Graph (CDFG). CDFG is then mapped by processing each of its basic blocs (*i.e.* DFG) sequentially. To keep the method with a low complexity, a list-scheduling based algorithm is used to schedule nodes of each DFG. As it is a local greedy method, the binding is made simultaneously to ensure that at least one solution exists, hence avoiding dead-ends. The proposed binding step is realized incrementally by using an exact method. However, as exact methods do not scale up, a stochastic selection is performed. This allows for keeping only a reasonable number of solutions among all the possible partial solutions that have been found. Finally DFGs are dynamically transformed as needed when no mapping solution can been found. For that purpose, DFG are backward traversed to allow applying the most relevant transformation.

A. Architecture and Application Modeling

CGRA is modeled by a bipartite directed graph with two types of nodes: operator and register, in which temporal aspect is implicitly represented by connections from registers to operators. Fig. 3(b) illustrates the model of the CGRA presented in Fig. 3a). Two subtypes of operator nodes are defined. The first one is conventional operator that represents the physical implementation of an operation (+, ×, -…). A conventional operator is usually able to compute different types of operations (*e.g.* + and -) and/or memory access (*e.g.* load/store). The second type of operator is memorization operator. It is associated to a register and represents the operation of keeping a value in a register explicitly. Connection between output register and conventional

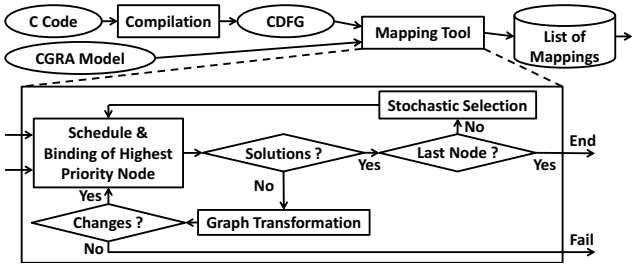

Fig. 2. General flow and algorithm core.

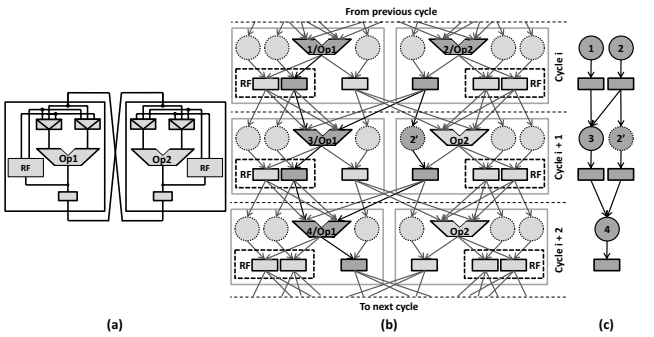

Fig. 3. **(a)** A 2×1 CGRA with 2 registers in RF, **(b)** Equivalent graph model on 3 cycles, **(c)** DFG model. In **(b)** a possible mapping of **(c)** is represented in dark grey. Memorization nodes are dotted and registers rectangular.

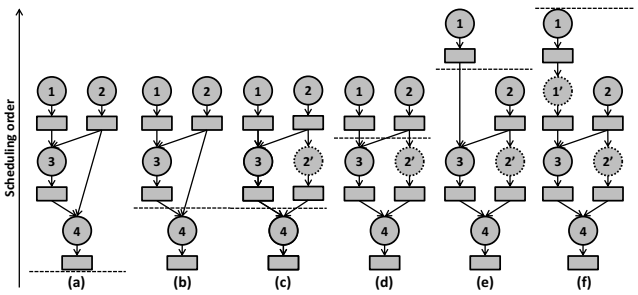

Fig. 4. Example of scheduled and transformed DFG on a one-tile CGRA. **(a)** Inital DFG, **(b)** after node 4 schedule, **(c)** after adding node 2', **(d)** after scheduling node 3 and 2', **(e)** after node 2 schedule, **(f)** Scheduled DFG after routing and scheduling node 1. Horizontal line shows the limit between scheduled and non scheduled nodes. Memorization nodes are doted circles.

operators depends on the interconnect network (*e.g.* in Fig. 3, only output registers can communicate with the operator of the other tile). Our CGRA graph model, that is very versatile, differs from the Time Extended CGRA proposed in [12] by explicitly representing registers. It can thus represent CGRA with homogeneous or heterogeneous tiles, presence or absence of RF, shared or local RF, homogeneous or heterogeneous operators, regular or specific interconnect network and operators that require more than one cycle to execute its operation (*e.g.* multipliers).

Application is modeled as a CDFG. CDFG is composed of a Control Flow Graph (CFG) and a set of basic blocks represented by DFGs. DFG is a bipartite directed acyclic graph composed of data nodes (rectangles in Fig. 3(c) and Fig. 4), operation nodes (circles) and data dependencies (arcs). In the proposed approach, in addition to conventional computation nodes (+, ×, -…), a particular operation node is introduced: *memorization*. The purpose of memorization node is to make data dependencies explicit along cycles. For example, 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. Memorization nodes are added when needed (*i.e.* when an operation has to be postponed) by graph transformations (section III.B.3).

Three equivalences between DFG and CGRA graph models nodes are defined: (1) data and register; (2) computation and conventional operator; (3) memorization operation and memorization operator.
As a result, the two models are homomorphic. Binding a DFG on a CGRA is therefore an equivalent problem to finding a DFG in the CGRA graph. This problem is known as the maximum common sub-graph problem and can be solved, as in [12] and [13], by using a method derived from Levi's algorithm [15].

B. Mapping Algorithm

As previously mentioned, the mapping algorithm is composed of four interdependent parts: scheduling, binding, graph transformation and a stochastic selection. These tasks are described in details in the next sub-sections.

1) Scheduling

The proposed approach uses a backward traversal list scheduling algorithm to schedule each DFG. It relies on a heuristic in which the schedulable operations are listed by priority order. In backward traversal, a node is schedulable if and only if all its children are already scheduled (*e.g.* node 2, in Fig. 4(b), is not schedulable since node 3 is not yet scheduled. So it has to be routed to keep data dependency resulting in Fig. 4(c)). The priority of nodes depends on their mobility, as defined in [18]. It is possible to process memorization nodes and conventional nodes differently. Also, when several nodes have the same mobility, their respective number of successors is used as a second priority criterion. The higher the number of successors, the higher the priority is. Indeed, a node with a higher number of successors is more difficult to map due to routing constraint coming from the limited amount of connections between tiles. Thus, scheduling these nodes at first usually allows enhancing the application's latency (*e.g.* node 2 in Fig. 4(d) has a higher priority than node 1). As soon as the highest priority node has been defined, our approach tries to find a binding solution. If a binding solution exists, the node is scheduled else the graph is transformed.

2) Binding

The proposed binding algorithm uses an *incremental* version of Levi's algorithm and thus differs from the one described in [12] that use the original version of Levi's algorithm *i.e.* fully exhaustive search of the whole DFG. Thereby, the algorithm we propose adds the newly scheduled operation node and its associated data node to the sub-graph composed of already scheduled and bound nodes. Only the previous set of solutions that have been kept are used to find every possibility to add this couple of nodes without considering the non-yet scheduled nodes. If no solution is found, there is absolutely no possibility to bind this couple in all the previous partial solutions because Levi's algorithm provides a complete exploration of the available solution space. In that case, graph transformation is required.

3) Graph Transformations

DFG is transformed *dynamically* when required, as opposed to existing works like [12] and [13] that apply static *a priori* transformations, or like [10] and [11] that apply *random*

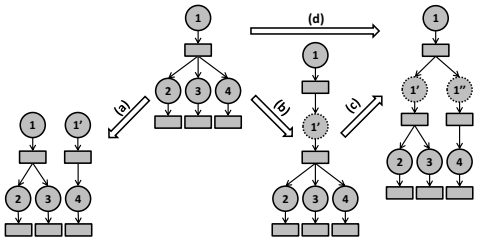

Fig. 5. DFG transformations: **(a)** Operation Splitting, **(b)** Simple Routing, **(c)** Memorization Spliting, **(d)** Routing & Splitting.

transformations. Fig. 5 presents a simple DFG and the results of the four different transformations.

- *"Operation Splitting"* duplicates an operation node by keeping its same inputs and distributing output edges to reduce the number of successors of the original operation node (see Fig. 5 (a)). "Operation Splitting" is equivalent to "recomputing" in [12].
- *"Simple Routing"* adds a memorization node and its associated data node to delay one operation and to keep data dependencies (see Fig. 5 (b)).
- *"Memorization Splitting"* (Fig. 5 (c)) is equivalent to "Operation Splitting" but applied to memorization nodes. It adds another memorization node with the same parents at the current cycle and distributes edges to reduce the number of successors of the original node.
- *"Routing & Splitting"* is the combination of "Simple Route" and "Memorization Splitting". It delays the schedule of an operation and reduces the number of successors of the generated memorization node (see Fig. 5(d)).

There are two situations where a graph transformation is required. The first one occurs when one of the parent operations of a previously scheduled node is not schedulable (*e.g.* node 2 in Fig. 4(b) which is one of node 3 parents). In that case, both "Simple Routing" and "Operation Splitting" are available. The choice is made by using a cost function which inputs are the number of parents, the free resources and the number of successors (*e.g.* in Fig. 4 (b), "Simple Routing" is used because the number of free resources is equal to the number of schedulable operations). The other situation occurs when the binding algorithm does not find any solution with a couple "operation – data". Two reasons are possible: there are either no more free operators left in the CGRA or the produced data cannot concurrently reach the already bound successors operations through the interconnection network. In the first case, "Simple Routing" is the only available transformation. In the other case, the four transformations are possible. The choice is also realized by using a cost function.

4) Stochastic Selection

The comprehensive aspect of Levi's algorithm can lead to a very large number of partial mappings (depending on the data dependencies and the architectural constraints) which can prevent its use with complex DFG and/or complex CGRA (*i.e.* a large number of nodes or tiles). A first idea to reduce this number is to remove redundant partial mappings. A partial mapping is redundant when it uses exactly the same operators to make the same operations than another partial mapping at the current scheduling cycle. This step allows for keeping only all the different partial solutions and preserving an exhaustive search. However, as illustrated in experiment results, this pruning technique does not scale well (see "Method 3").

To keep both computation time and memory usage to a reasonable level in the mapping tool, we propose to use a stochastic selection instead of removing redundant partial mappings. This pruning step is made after the binding step and before the scheduling of the next node (as illustrated in Fig. 2). Let the result of the binding be a list of *nbMappings* partial solutions. For each partial mapping, a random number between 0 and 1 is randomly generated and compared to a threshold. This threshold has to be chosen carefully: it should be low enough to scale up and high enough to allow for finding a good solution. Unfortunately, as shown in the experiments, using fixed value as threshold (*e.g.* 75% of *nbMappings*) gives very bad quality results: depending on the DFG, it sometimes has a very high failure rate, sometimes a non reasonable computation time and that for a large number of threshold values. Thus, the threshold should adapt itself to *nbMappings*. For that purpose, *nbMappings* is normalized by a reference number λ set by the user and this number is used by the threshold function. This function should have the following characteristics: tend to 1 when *nbMappings*/λ is small and be a decreasing function. Many functions can be considered (*e.g.* exponential, invert, hyperbolic tangent, etc.). To define the acceptance threshold, we chose a decreasing exponential that is classically used in simulated annealing (see Equation 1)

$$Threshold(nbMappings, \lambda) = \exp\left(\frac{-nbMappings}{\lambda}\right) \qquad (1)$$

Once all the nodes of the application have been scheduled and bound, our method selects one of the best solutions (*i.e.* a solution with the lowest latency) among all the available mappings.

IV. Experiments and Results

A. Experimental Setup

The proposed synthesis flow has been fully automated through a software tool implemented in. GCC 4.7 has been derived to generate CDFGs from applications described in C language. Nine applications from signal processing domain and High Level Synthesis (HLS) benchmarks have been used for our experiments, namely: 2D Discrete Cosine Transform (DCT-2D), matrix product, Fast Fourier Transform (FFT), Manhattan Distance computation, Exponential Moving Average Filter (EMA), Moving Window De-convolution (MWD), Unsharp Mask, Elliptic Filter and a Low-Pass filter (DC Filter). A workstation integrating an Intel Xeon and 8 GB of RAM has been used to realize the experiments.

Firstly, the impact of the user-defined parameter λ is studied to determine the best compromise according to the metrics. For that purpose, λ varies in {1000, 2500, 5000, 10000, 25000, 50000}.

Then the proposed approach with proper λ is compared to state-of-the-art approaches. As a reminder, our method backward traverses the graph, schedules and binds nodes simultaneously with dynamic transformations and

stochastically selects partial mappings to prune the solution space. Since [12] and [13] have been shown to provide better results than [8] and [11], we do not compare with these approaches in this paper. The three other methods we have considered for comparison are:

- *"Method 1"* that solves the scheduling and the binding problem separately as proposed to generate in [7]. Graphs are transformed during scheduling by applying "Simple Route" transformation only. A forward list-based scheduling algorithm and the original Levi's binding algorithm are used.
- *"Method 2"* that traverses the graph forwardly, schedules nodes by applying *a priori* transformations and tries to find a mapping by using the original Levi's algorithm as proposed in [12], [13].
- *"Method 3"* that backward traverses the graph, schedules and binds nodes simultaneously with dynamic transformations as proposed in this paper and removes redundant partial mappings to prune the solution space.

To obtain a large spectrum of results, several constraints have been varied: CGRA size, RF size and the number of tiles the final mapping is allowed to use. This has led to 16 different sets of constraints per application and per method for which computation time is reasonable (under 1000s). Also, since proposed method is based on stochastic selection, we present average results based on ten executions.

Several metrics have been considered to assess quality of the methods. They can be grouped into three main categories. The first category assesses the intrinsic quality of a mapping method. It is composed of two metrics: (1) the success rate, defined as the percentage of time a method finds a mapping when at least one of the four method succeed, and (2) the percentage of time the method gives the best latency between the four methods. The second category illustrates the ability of a method to widely and efficiently explore the design space. For that purpose, two criteria are considered: the number of different mappings found and the number of different mappings found per second (throughput). Two mappings are different if they use different tiles or use the interconnection network differently. These two metrics really illustrate the ability of a method to transform the application graph just as needed and so to efficiently explore the solution space. Indeed, adding more nodes than needed reduces the number of different mappings found at the end because it over constraints the problem. The third category gives the ability of a method to scale up. The underlying metric is the computation time.

B. Results

TABLE 1 presents the influence of λ over success rate and the number of different mappings generated per second ("Mappings throughput") because the impacts on the other metrics are not indicative. λ has an influence on the success rate below 25000. Above 5000, this influence is less than 1%. "Mappings throughput" observation shows that there is a value of λ that maximizes the number of different mappings which is near 10000. Accordingly, for the rest of the experiments, λ is set to 10000 for the proposed approach. For each previously defined criterion, Fig. 6 to Fig. 11 illustrate results of the four methods.

Fig. 6 shows that Method 1, that solves scheduling and binding totally independently, leads to the lowest success rate (almost 40%). Method 2, that transforms the graph a priori, provides better results (~65%) but is not as good as Method 3 (~90%) and the proposed approach (~97%). For some of the benchmarks (*e.g.* unsharp mask), our stochastic selection is even better than the exhaustive method because the partial solutions found become dead-ends and that the stochastic

TABLE 1 LAMBDA INFLUENCE

λ value	1000	2500	5000	10000	25000	50000
Success Rate Decrease	3.7%	1.85%	0.46%	0.92%	0%	0%
Mappings Throughput	0.47	0.60	0.73	0.80	0.77	0.64

selection removes them. Indeed, the exhaustive search tries to transform the graph as less as possible. It keeps many partial solutions that lead to dead-end and has not enough time to finish the exploration in the given time-out. The stochastic approach can discard the solutions that lead to dead-ends and save time to completely explore further (and possibly transform) the partial solutions kept.

When a method finds the solution, we evaluate the quality of the result by comparing to the ASAP length of the DFG. Fig. 7 shows the percentage of time the method finds the best latency (when it finds one). The proposed approach is based on a backward list-scheduling which is a heuristic and so cannot always find the best latency. However, as shown in Fig. 7, our method finds most of the time the best latency (83%). Moreover, the random aspect of the stochastic selection sometimes allows for improvement compared to Method 3 which is exhaustive (but not exact). Indeed, it may change the schedule which can result in an improvement of the latency (diversification well-known phenomenon).

In Fig. 8, the average number of different mappings generated is presented. It shows that adding stochastic selection does not really decrease the diversity of the results

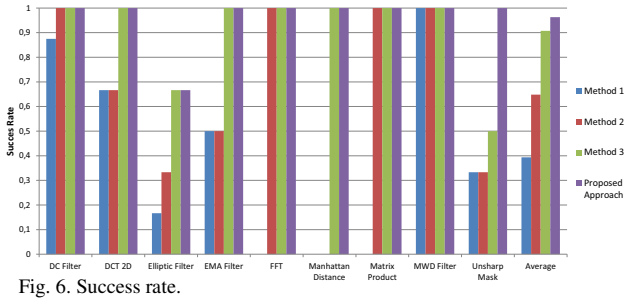

Fig. 6. Success rate.

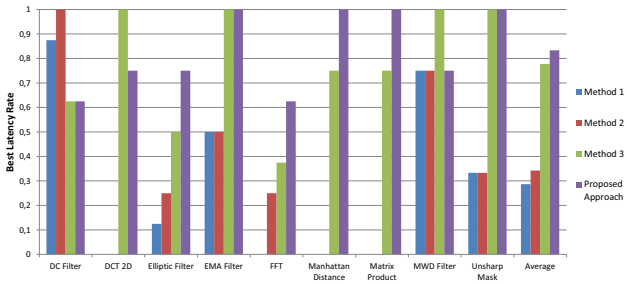

Fig. 7. Percentages of time methods find the best latency.

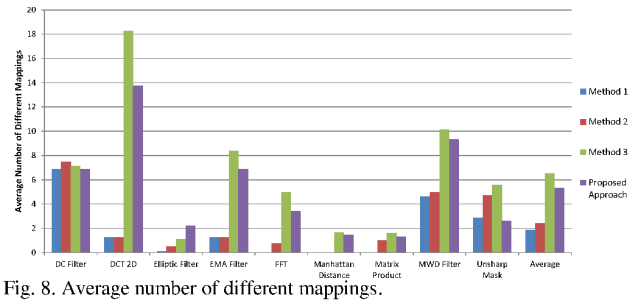

Fig. 8. Average number of different mappings.

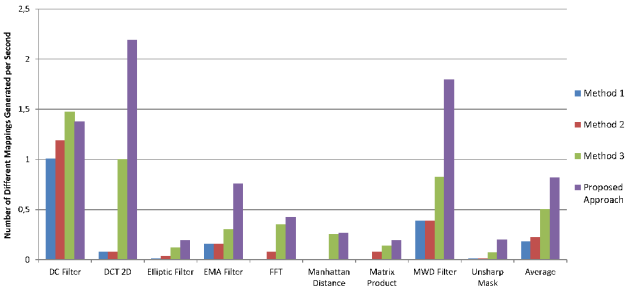

Fig. 9. Average number of different mappings generated per second.

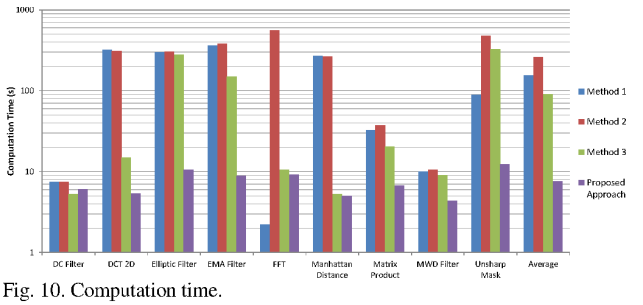

Fig. 10. Computation time.

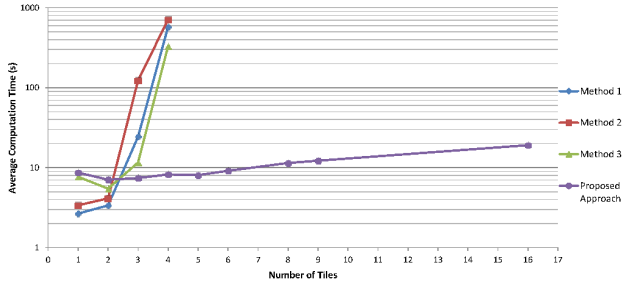

Fig. 11. Average computation time for different number of tiles.

(and so the quality of the exploration) compared to the exact method (~18% less). Methods 1 and 2, whose primary goal is not diversity, do not work as well as the two other ones.

In Fig. 9, the average number of different mappings generated per second shows that proposed method achieves respectively 4.5, 3.7 and 1.6 times better mappings throughput and thus has higher exploration efficiency by spending less time finding more different mappings than the other methods.

Fig. 10 shows that, most of the time, the proposed approach has a lower computation time than the other methods. Furthermore, Fig. 11 illustrates that our method is more scalable. Indeed, for Methods 1, 2 and 3 computation times evolve exponentially on a logarithmic scale and become prohibitive ($\gtrsim$1000 seconds) when considering more than four tiles. With our approach, it slowly increases with the number of tiles and is under 20 seconds even with 16 tiles.

V. Conclusion

In this paper, a generic method to map applications written in high level language on CGRA is presented. The proposed approach leverages on simultaneous scheduling and binding steps respectively based on a heuristic and an exact method stochastically degenerated. The formal graph model of the application, obtained after compilation, is backward traversed and dynamically transformed to allow for a better exploration of the design space. Experimental results show that this method provides very good results and is scalable.

References

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

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

[3] 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.

[4] 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 *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, 2007.

[5] S. Pillement, O. Sentieys, and R. David, "DART: A Functional-Level Reconfigurable Architecture for High Energy Efficiency," *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 1–13, 2008.

[6] 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 *Field Programmable Logic and Application*, P. Y. K. Cheung and G. Constantinides, Eds. Springer Berlin / Heidelberg, 2003, pp. 61–70.

[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 *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008.

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

[10] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "DRESC: A retargetable compiler for coarse-grained reconfigurable architectures," in *Field-Programmable Technology, 2002. (FPT). IEEE International Conference on*, 2002, pp. 166–173.

[11] 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.

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

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

[14] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proceedings of the 27th annual international symposium on Microarchitecture*, 1994, pp. 63–74.

[15] 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.

[16] 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 *Field Programmable Logic and Applications, International Conference on*, 2006.

[17] 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 *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2010, pp. 168–175.

[18] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 8, no. 6, pp. 661–679, 1989.

[19] L. Chen et T. Mitra, « Graph Minor Approach for Application Mapping on CGRAs », ACM Trans Reconfigurable Technol Syst, vol. 7, no 3, p. 21:1–21:25, sept. 2014.