Twenty Years of Automated Methods for Mapping Applications on CGRA

Kevin J M Martin

Twenty Years of Automated Methods for Mapping Applications on CGRA

Kevin J. M. Martin
Université de Bretagne Sud
Lab-STICC, UMR CNRS 6285
Lorient, France
kevin.martin@univ-ubs.fr

***Abstract*—Coarse-Grained Reconfigurable Architectures (CGRAs) emerged about 30 years ago. The very first CGRAs were programmed manually. Fortunately, some compilation approaches appeared rapidly to automate the mapping process. Numerous surveys on these architectures exist. Other surveys also gather the tools and methods, but none of them focuses on the mapping process only. This paper focuses solely on automated methods and techniques for mapping applications on CGRA and covers the last two decades of research. This paper aims at providing the terminology, the problem formulation, and a classification of existing methods. The paper ends with research challenges and trends for the future.**

***Index Terms*—CGRA, Mapping, Compilation**

I. Introduction

Despite three decades of constant study, Coarse Grain Reconfigurable Architectures (CGRAs) are still in 2022 the ever promising solution that did not yet meet the expected commercial success. Computer architecture is entering a new golden age [1] and CGRAs might eventually go beyond promise. CGRAs are seen as good compromise between the necessary flexibility and computing power needed by next-generation applications and the energy-efficiency required by all systems, not only the embedded ones. CGRAs gather together a huge set of possible architectures, ranging from simple organisations to complex ones [2], [3]. One may even consider also GPGPUs as part of this big family [4]. Indeed, the design space is huge and includes several architectural dimensions: processing elements and their homogeneity, interconnection network, context frame, partial reconfiguration, orchestration mechanism, design of memory hierarchy, and host-CGRA coupling to name a few. The number of proposed

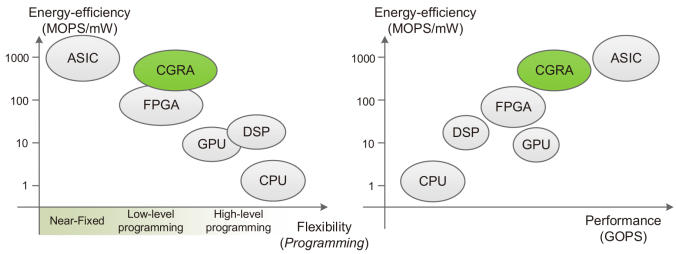

Fig. 1. Architecture comparison proposed in [3]

architectures is simply tremendous and undoubtedly, CGRAs still keep a wide unexplored area. From its reconfigurable features, CGRAs are a key member of the reconfigurable computing family. Figure 1 shows the ideal trade-off between flexibility, performance, and energy efficiency that CGRAs offer compaired with other architectures.

Making an inventory of existing CGRAs is a complex and time consuming task that has been successfully done in the past [2], [3], [5]–[8]. Fortunately, the newcomer in the domain can restrict to reading only few papers to acquire a nice overview. The Hartenstein's paper surveys the first decade of reconfigurable computing [2]. In 2010, De Sutter et al. published a book chapter detailing the architecture features of a CGRA [9]. Wijtvliet et al. review 25 years of CGRAs in 2016 [7]. The most recent surveys are provided by Liu et al. [3] and Podebas et al. [8]. Liu et al. [3] suggest another classification, complementary to the ones proposed in the previous surveys. Podebas et al. interestingly gather the published CGRAs from a performance perspective [8], and highlights by figures what is commonly accepted: CGRAs are serious competitors to GPGPUs[1]. These two last surveys point out the severe limitations that CGRAs meet like the unadapted programming model.

The abovementioned papers focus on the architectures. In order to make use of the abondant number of processing elements available, a CGRA must come along with a compiler. The very early CGRAs were programmed manually, i.e. at assembly level [10]. These first steps were important to understand how to program such an architecture and describe a systematic method that can then be automated. The automated

[1]provided that we do not consider GPGPUs as part of the big CGRA family

process of programming a CGRA from a high level language falls in the compilation category. The backend part, responsible for defining the use of the hardware resources is called application mapping.

An inventory of existing mapping techniques has also been done in the past [5], [6], [11]. Theodoridis et al. present the CAD tools along with the CGRAs up to 2007 [5]. In 2011, Choi [6] wrote a survey that combines both architecture and application mapping. These papers present first the architecture, and their associated mapping flow individually. In [11], a survey on compiling for reconfigurable computing architectures covers the broad range of reconfigurable computing, including FPGAs, up to 2010. The common features in the compiler are described, and then some dedicated compilers are presented. This paper focuses on automated methods for mapping on CGRAs only, includes the last decade of research on that topic, and proposes a classification. From the early first papers [12]–[14] to the latest publications on the topic [15]–[17], this paper paints a picture of two decades of CGRA mapping.

Before presenting what is the mapping problem and the techniques to solve it, this survey proposes a terminology to clearly state the problem, and extracts a general problem formulation. This paper concludes with the research challenges to be taken up.

II. Terminology and Methodology

This paper starts with definitions and terminology, that might not be so obvious even for experts of the topic. Defining the terms allows for a newcomer to get familiar with the terms.

A. What a CGRA looks like

Even if this paper focuses on the mapping technique, it presents a typical CGRA and describes its main architectural features, which are essential to introduce in order to understand the mapping problem. Figure 2, taken from [8], presents a simple CGRA which contains the minimal components of a basic CGRA. A CGRA is a set of processing elements (PEs), also called reconfigurable cells (RCs), or *tile*, or functional unit (FU). The term cell might be more generic, as in some CGRAs, the cells are heterogeneous, composed of computation unit, or memory units. This set of cells is usually placed as a two dimensions array, where the cells are interconnected through point to point connections, or more complex topologies. The interested reader is invited to read the dedicated papers for more details [3], [5], [8], [18]. The key element to highlight is that a CGRA exposes both spatial and temporal parallelism.

B. Terminology and definitions

Array or Architecture? A first ambiguity is the 'A' of CGRA. In the literature, it sometimes stands for "Array", sometimes for "Architecture". Both cases make sense. A CGRA is typically organized around an *array* of cells, but the word *architecture* encompasses all kind of organisations, not only array-based.

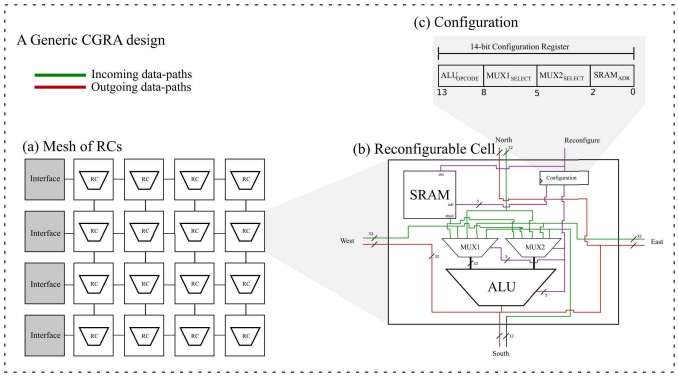

Fig. 2. Illustration of a simple CGRA taken from [8], showing the mesh topology (a), the internal architecture of the Reconfigurable Cell, RC (b), and an example of the configuration register (c).

Why reconfigurable? A CGRA is a *reconfigurable* architecture. As such, it relies on *configurations*. The term *context* or *control* are also commonly found in the literature to mean a configuration. Some authors may even use the term *instruction*. A newcomer might wonder what is the difference between a *configuration*, a *context*, and an *instruction* of a CGRA. The difference lies in the hardware that allows to reconfigure the architecture. Finally, in all CGRAs, the reconfiguration is a matter of signals that drive multiplexers in a data-path. Therefore, a *configuration* must hold all the values of a set of signals that select the correct input of a multiplexer. A *context* is such a structure that contains all the *raw* values. An *instruction* can be seen as a *condensed* representation of a context. An instruction needs to go through a decoder whose outputs drive the multiplexers. Deducing that a processor is a reconfigurable architecture is a precocious conclusion that we cannot draw though. But whether it be a *context* or an *instruction*, the importance from the compilation point of view is to know what to produce as the format defines the contract between the hardware and the software to reach a valid execution.

Spatial computation vs. temporal computation. One of the crucial hardware feature that the compiler must *know* is if the CGRA supports spatial computations or temporal computations [3]. Spatial computation is very similar to FPGAs. Along with spatial computations that all CGRAs support, temporal computations allow to share in time the hardware resources leading to more flexibility, but are often criticized to reduce the energy efficiency [19].

Compilation. *Compilation* is an automated process that takes an input source code and transforms it into an equivalent *binary* code, executable by a given architecture. Fig. 3 shows a typical compilation flow for CGRAs. A compiler is conceptually composed of three main steps: (1) the front-end, in charge of parsing the source code and producing an equivalent intermediate representation (IR), (2) the middle-end, where some optimization passes may occur on the IR[2], and (3) the back-end, responsible for producing the binary code from

[2]in real life there are multiple IRs according to the optimisation to perform

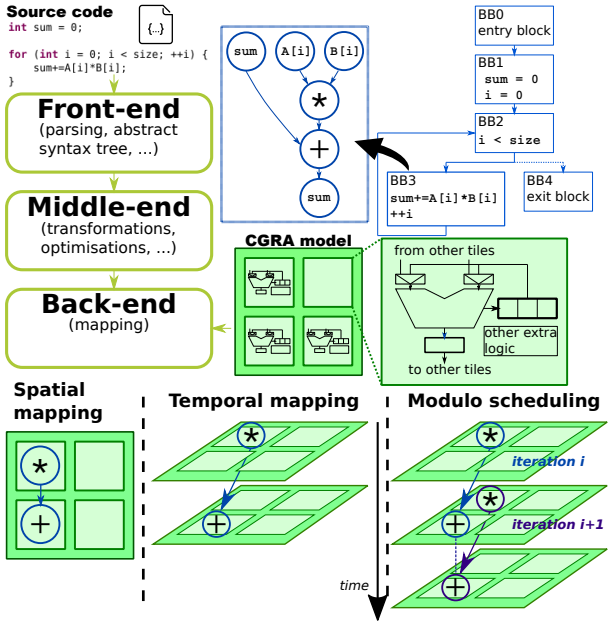

Fig. 3. Classical compilation flow for CGRAs

the IR. Thus the back-end must *know* the target architecture. The specific features of the early CGRAs were *hardcoded* in their own compiler, some techniques being specific to a very particular hardware and hardly reusable. This is why the previous surveys present individually the compilers [5], [6], [11], as they are all tailored to a specific target. Designing a *retargetable* compiler for CGRAs is still an open issue today.

DFG, CDFG. The intermediate representation of a compiler is usually in the form of a graph. A Data Flow Graph (DFG) is a graph whose nodes represent operations and whose edges are the data dependencies between the operations. A DFG is embedded in a basic block, such that a basic block has a single entry and single exit. Fig. 3 shows an example of a DFG inside a basic block (BB3). A Control Flow Graph is a graph whose nodes are basic blocks and whose edges are the control dependencies between the basic blocks. The combination of the two forms a CDFG (Control Data Flow Graph). An application specified in a given language can thus be represented in the form of a graph, where the nodes are the operations, and the edges are the dependencies (control or data).

Binding or placing? The word *binding* holds the idea to tie things together, whereas *placing* let think a little freedom about the spatial location. Both terms are equally used in the literature for the same meaning. We choose *binding* for the rest of this paper.

Routing. The routing defines the physical connections between the computing resources. Routing usually builds on *placement*. In a CGRA, placement and routing of computing resources are already done. In that context, routing does not mean creating a new route with a physical wire, but use an existing link without interfering with already existing communications using this link.

Scheduling. The scheduling is the method that assigns *in time*, operations to the hardware resource, while guaranteeing the dependencies.

Mapping. The *mapping* in the main step in the back-end. The word *mapping* can designate both the process and the output of the process. For a spatial CGRA, the mapping process amounts to solving the binding problem. For temporal CGRA, the mapping process must solve both binding and scheduling problems. When the problem is solved, the output of the process is a *valid mapping*, i.e. a binding (and scheduling) of operations of the application on the hardware resources while guaranteeing the dependencies. Fig. 3 shows a *spatial mapping* and a *temporal mapping* of a simple dot-product input source code. Spatial mapping is also sometimes referred to as *straight forward mapping*.

Software pipelining and modulo scheduling. *Software pipelining* is a general technique for overlapping loop iterations. *Modulo scheduling* is the most commonly used technique for software pipelining, especially in the CGRA domain. In [20], the authors define clearly the goal: "*The objective of modulo-scheduling is to engineer a schedule for one iteration of a loop such that this same schedule can be initiated at regular, as short as possible, intervals, taking into account data dependences and resource constraints. This interval in terms of cycles is termed initiation interval (II)*". The quest of the minimum II is the main motivation of many works. Fig. 3 shows an example of modulo scheduling for the dot-product. The II in the example is one, and the figure clearly shows that two different iterations of the loop are being processed at the same time.

CGRA models. The back-end must *know* the target architecture in order to fully make use of its specific features. The idea of *describing* to the compiler the CGRA emerged very early [21]. In 2002, Mei et al. [22] use an architecture abstraction in the famous DRESC. In 2003, Lee et al. propose a generic architecture template called the dynamically reconfigurable ALU array (DRAA) [14]. Since then, the great majority of works considers a model of the CGRA as input of the compilation flow. Recent efforts include non-functional constraints [18].

C. Problem formulation

This paper focuses on methods to solve the mapping problem, which combines two NP-complete problems: scheduling and binding. This raises CGRA compilation as a unique scientifique problem and main challenge, because mapping might fail [23]–[25], which is of course unconceivable from the user point of view. To this end, for instance, HiMap [26] is an iterative algorithm that terminates when a valid mapping is found. Historically, CGRA mapping is the meeting point between VLIW compilation, and FPGA place-and-route. The difference with VLIW compilation is the direct communication possibilities offered by the CGRAs between the different PEs. VLIW processors share data through a register file only. The difference with FPGA place-and-route is the granularity of the processing elements and a usually less flexible interconnect. A

single formalisation of the mapping problem is not possible, as it is specific to the architecture model and the execution model considered. The interested reader can refer to other papers where the authors clearly formalized their problem [23], [27]. The idea is to give an explanation of the problem, understandable by a newcomer.

A nice definition is given in [3]: "*the mapping of a CGRA is actually equivalent to identifying the spatial and temporal coordinates of every node and arc in the control/data flow graph (CDFG). Compilers are responsible for making this arrangement.*" We may add that the temporal coordinate system is often called the *time extended CGRA* (TEC) [28], or the *time-space graph* [29]. The challenge is reminded by Chen et al. [27]: to provide *high quality solution with fast compilation time*. Thus, the mapping problem can be summarized as follows: bind in place and schedule in time operations of the application on the CGRA while guaranteeing the dependencies and in a short time, such that the application executes as fast as possible.

D. Methodology

This paper is solely based on the information found in the publications available online. The authors of the cited papers have not been contacted for more thorough explanation about the techniques (the lack of space when writing papers forces sometimes the authors to simplify or omit some aspects), and the personal knowledge of the authors of this paper is not used.

The papers cited in this survey have been published in high ranked international conferences or journals. As the ranking is an endless debate, and the ranking is subject to change, the international audience targeted preveals. National conferences or journals, even when the proceedings are written in English, are rarely considered for this survey. There are several strong places of CGRAs all around the world. Some groups have a long history and contributed heavily to the increase of knowledge about the CGRAs and corresponding mapping techniques. Only a subset of papers is cited in this survey.

III. A review of mapping methods

This section presents a round-trip of proposed methods to solving the CGRA mapping problem. As the application is composed of data-flow parts, and control-flow parts, some methods have been devised specifically for each part. The section ends with an overview of the scientific production of the last two decades.

A. Data-flow mapping

All the works cited in this paper propose a technique to map the data-flow part of an application. Some methods follow a place-and-route similar to what is done in FPGAs. Some other methods formalized the problem to delegate to a solver. Since the mapping problem is a NP-complete problem, researchers naturally looked after techniques provided by the operational research or graph theory domains. These have been extensively used to solve the data-flow mapping problem.

In [26], Wijerathne et al. suggest to classify the existing CGRA mapping algorithms into three main categories: heuristic-based, graph-based or ILP-based (Integer-Linear Programming). The graph-based approach can be discussed as soon as all applications are modeled by graph, but graph-based approaches use techniques borrowed from the graph theory domain and can deserve an own class. For instance, Hamzeh et al. [46] build a compatibility graph, Dave et al. rely on the max clique in RAMP [38], a graph minor approach is followed in [27], and the maximum common subgraph is used in [28], [47], [54]. We suggest to follow a more usual way of classifying optimization algorithms between approximate methods and exact methods. The other issue with graph-based approaches, in the general case, is that some of them can be exact and others not. Specifically for the case of CGRA mapping, it appears that all of them are heuristics. We also suggest to further divide the heuristic-based approaches by adding a meta-heuristic category (which is part of the heuristics category but this makes Table I more clear). Finally, we include the ILP into an "exact-based" category, which is also debatable as soon as these techniques do not guarantee an optimal mapping. The main feature of the exact based methods is that they can prove the optimality, whereas heuristics may find the optimal solution, but without the possibility to prove it.

As a second dimension to this classification, we suggest to differentiate spatial and temporal mapping. In the case of temporal mapping, the binding and scheduling steps can be solved together, or can be solved separately (one after the other). In that case, some approaches may use a combination of techniques. Table I gathers all the techniques used to solve the mapping problem, for spatial or temporal architectures. The different techniques used are presented in four main columns: (1) heuristics, (2) meta-heuristics, (3) ILP or Branch and Bound (B&B) methods, (4) Constraint Satisfaction Problems (CSP). The heuristics encompass all the techniques specifically designed for the given problem. The meta-heuristics form a family of optimisation algorithms, and the table further divides it into two families: population-based technics like Genetic Algorithms (GA) or quantum-inspired evolutionary algorithm (QEA), and local search techniques like Simulated Annealing (SA). The exact methods include Integer Linear Programming (ILP) and branch and bound on one side, and techniques that model the mapping problem as a constraint satisfaction problem. This problem is then solved through constraint programming (CP), SAT (Boolean satisfiability), or SMT (Satisfiability Modulo Theories). Please note that all the papers cited do not appear in the table, as some of them rely on already referenced papers. For instance, the approach presented by De Sutter et al. [20] relies on DRESC compiler [22], which already appears in the table.

B. Control-flow mapping

Mapping the control-flow graph raises another difficulty. A solution adopted in many cases is to let the control flow managed by a host processor. But this reduces greatly the pos-

TABLE I
A REVIEW OF BINDING AND SCHEDULING TECHNIQUES FOR AUTOMATED SPATIAL AND TEMPORAL MAPPING OF APPLICATIONS ON CGRAS.

	Approximate methods			Exact methods	
	Heuristics	Meta-heuristics		ILP/B&B	CSP
		Population-based	local search		
Spatial mapping	[23], [30], [31]	GA [19]	SA [32], [33]	ILP [23], [34], [35]	
Temporal mapping	[12], [16], [26], [36]–[40]		SA [22]	ILP [41] B&B [42]	CP [43] SAT [17] SMT [44]
Binding	[14], [24], [28], [45]–[47]	QEA [48]	SA [30], [49], [50]	ILP [15], [48]	
Scheduling	[24], [28], [36], [46], [48], [50]–[52]			ILP [15], [53]	

sibilities to use the CGRA and increases the communication overhead, loosing sometimes the benefit of the acceleration provided by the CGRA. Another approach is to provide the CGRA with extra hardware features to support the control flow. Two structures are distinguished: Conditional and alternative structures, and iterative structures.

1) Conditional and alternative structures: Conditional and alternative structures are if-then-else (ITE) constructs. As clearly presented in [55], there are four basic methods to map applications with ITE onto CGRAs: (1) Full predication [56], (2) Partial predication [57], (3) Dual-issue single execution [55], [58], [59], (4) Direct CDFG mapping [60]. Supporting ITE constructs efficiently is still a hot topic, as witnessed by recent publications [55], [59].

2) Iterative structures: Iterative structures are defined by an initialisation phase, an iteration condition, and an iteration step. Most of the works focus on *for loops*. Loops have been the primary care since the early days of CGRAs [12]. Since loops concentrate the most important computing part of the application, researches naturally focused on this specific case, and the topic has been intensively studied during the last two decades. Most of the works consider the loop body, letting the control flow managed by a host processor. When the loop body contains conditional or alternative structures, the techniques presented in III-B1 can be used. Mapping loops on CGRA is so intensively studied that it would certainly deserve a survey on its own.

Modulo scheduling. Modulo scheduling is the most widely used technique to map loops on the CGRA [29], [30], [52], [61]. It can rely on a modulo routing resource graph (MRRG) [59], [61]. It can also be solved through graph-based approaches [37], [38].

Hardware loops. Hardware loops consist of extra logic inside the CGRA to manage the iterations of the loop in order to reduce the overhead of loop control by the processor [62]–[64].

C. Data mapping

The interaction between the CGRA and the memory is also of utmost importance as it defines the efficiency of the whole execution of the application. Various parameters of the memory can be considered for an efficient mapping: number of banks, communication bandwidth, and memory size [50], [65]–[68].

The internal memory resources of the CGRA should also be used efficiently. Register allocation is presented in [29], [46], for a rotating register file [29], or for a unified register file [25].

D. Timeline

Fig. 4 presents the evolution of scientific production around CGRA mapping the last two decades. The number of publications per year is not accurate, as it considers the papers focusing on CGRA mapping only, and a subset of selected papers, but still it shows that the community has intensified the efforts in the last decade, with a clear increase in 2021. The figure also shows that modulo scheduling was considered since the beginning of the studies, that supporting branches started in the early 2000s, and that memory-aware methods gained interest around 2010.

IV. THE FUTURE

The first wave of CGRA was fueled by signal processing applications, especially multimedia applications like image, audio, and video, for embedded systems, constrained by stringent power and energy budget. The Samsung Reconfigurable Processor (SRP) [69], an ADRES-like CGRA, integrated in the past in the Exynos SoC, is an example of a commercial use of CGRAs. The choice of Samsung to discontinue the use of SRP in favor of more conventional processors is the sign of a mitigated success [70].

A. Trends

CGRAs experience a new momentum as they get carried away by artificial intelligence (AI) applications. The massive need of high-performance computing, coupled with the slowdown of Moore's law and end of Dennard scaling, and the mismatch between AI workloads and conventional Von Neumann architectures, drives the efforts towards a multitude of AI-accelerators, which fall in the category of CGRAs. The diversity of names in the literature also shows that the domain is buzzing: Xilinx AI-engine [71], Reconfigurable Dataflow Architecture [72], Reconfigurable Dataflow Accelerator [73]. These "modern" CGRAs differ from the legacy ones in the number of cells that are available, which causes a serious scalability issue that is discussed in the challenge section. Another difference is the coupling with a host CPU. Modern CGRAs tend to be standalone, similarly to a GPU, and they require a full system integration. CGRAs are the relevant (if not the only credible) solution to take up the challenge of energy-efficient AI applications. The second wave of CGRAs might eventually be the one that meets an industrial success.

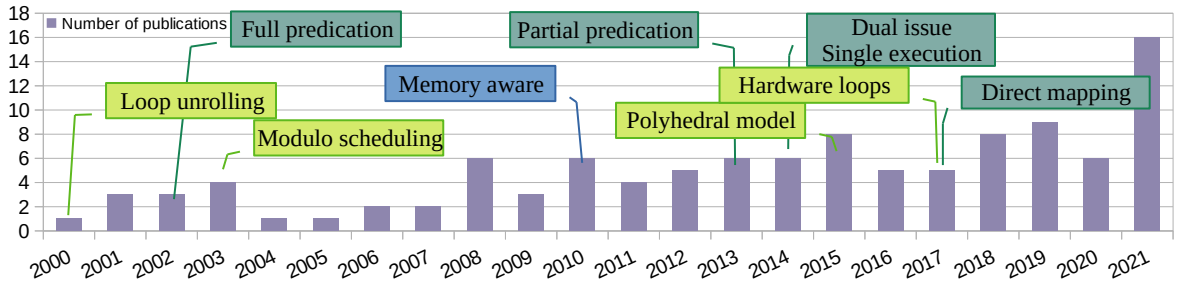

Fig. 4. Number of publications related to CGRA mapping over the last two decades. Note that this timeline does not include papers about CGRA architectures, and is not comprehensive.

The mapping problem is complex, and needs sophisticated algorithms that are time consuming to understand, and to formalize. The methods based on artificial intelligence and machine learning are clearly interesting trails [74]. AI-based methods help in concentrating within a single model some functional but also non functional contraints, that are hard to formalize through traditional methods. AI for electronic design automation in general is at its early beginnings, and AI for CGRA specifically will be part of this global trend.

From the architectural point of view, some evolutions will obviously impact the compilation. The CGRA coupling can also be further explored: near the memory, or directly integrated within the memory array in a *processing-in-memory* manner [3]. The emerging memory technologies will also be game changers.

Finally, some open-source frameworks recently appeared [32], [75]–[77] to share the technical efforts and provide a ready for use tool to democratize the CGRAs and make them widely adopted for energy-efficient or high-performance computing. These frameworks are also part of a wider trend about open source hardware.

B. Challenges

a) Programming model: The unadapted programming model used up to now for CGRAs is the main limitation identified in two recent surveys [3], [8]. Other programming models needs to be considered, more adapted to CGRAs, able to specify the data-level parallelism, like OpenMP, SYCL, CUDA or OpenCL. The dataflow model of computation could also be interesting to look at. This kind of streaming model can fit with CGRAs [31], [78].

b) Scalability: Scalability is clearly one of the biggest challenge to be taken up. Some techniques are already proposed do deal with scalability. In [26], the repetitive patterns of loops are detected and are mapped in a hierarchical way. In [24], the partial solutions are stochastically pruned to keep under control their number. But while legacy CGRAs are composed of tens of cells, with a use rate quite low limited by the intruction level parallelism available in the applications, the more recent and modern CGRAs, the most capable of crunching AI workloads, contains hundreds to thousands of cells. The issue is to effectively make use of the massive number of cells. The standalone feature of modern CGRAs is another game changer for mapping methods. The mapping problem is intractable, scalability further raises the challenge, and the number of cells involved takes it in yet another dimension. The application should be considered as a whole, not with intensive kernels to be offload to the CGRA and letting the host processor interact with the system. A holistic approach is thus needed to first analyse the input application, and then relevantly partition it for finally an efficient complete mapping. SARA [73] is such a recent approach. It relies on a hierarchical pipelining to further extract parallelism. For instance, at loop level, two sibling loops might be executed in parallel. When there are data dependencies across the loops, the memory consistency is managed by the compiler, and the instructions are ordered to guarantee the correct execution. SARA makes use of spatial parallelism and temporal parallelism. The iterations of the loops are overlapped at all levels as an advanced implementation of software pipelining (not only modulo scheduling). In other words, the new generation of compilers for CGRAs must be able to make use of all levels of parallelism : instruction level, data level, and loop level.

V. Conclusion

This paper presents twenty years of methods for mapping application on CGRA. It provides the terminology and basic knowledge for the newcomer. The paper focuses on offline methods from imperative language, presents a classification of the methods and shows a timeline of the last two decades. The paper ends with current trends, calls for open-source frameworks for a democratization of the CGRA technology, and discusses the challenges for the near future of modern CGRAs. Among the upcoming challenges, the use of other programming models, and scalability for modern CGRAs running AI workloads, are specifically identified.

References

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, p. 48–60, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3282307

[2] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '01. IEEE Press, 2001, p. 642–649.

[3] 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 Comput. Surv.*, vol. 52, no. 6, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3357375

[4] D. Voitsechov, O. Port, and Y. Etsion, "Inter-thread communication in multithreaded, reconfigurable coarse-grain arrays," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 42–54.

[5] G. Theodoridis, D. Soudris, and S. Vassiliadis, *A Survey of Coarse-Grain Reconfigurable Architectures and Cad Tools*. Dordrecht: Springer Netherlands, 2007, pp. 89–149. [Online]. Available: https://doi.org/10.1007/978-1-4020-6505-7_2

[6] K. Choi, "Coarse-grained reconfigurable array: Architecture and application mapping," *IPSJ Transactions on System LSI Design Methodology*, vol. 4, pp. 31–46, 2011.

[7] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2016, pp. 235–244.

[8] A. Podobas, K. Sano, and S. Matsuoka, "A survey on coarse-grained reconfigurable architectures from a performance perspective," *IEEE Access*, vol. 8, pp. 146 719–146 743, 2020.

[9] B. De Sutter, P. Raghavan, and A. Lambrechts, *Coarse-Grained Reconfigurable Array Architectures*. Boston, MA: Springer US, 2010, pp. 449–484. [Online]. Available: https://doi.org/10.1007/978-1-4419-6345-1_17

[10] P. M. Heysters and G. J. M. Smit, "Mapping of dsp algorithms on the montium architecture," in *Proceedings International Parallel and Distributed Processing Symposium*, 2003, pp. 6 pp.–.

[11] J. a. M. P. Cardoso, P. C. Diniz, and M. Weinhardt, "Compiling for reconfigurable computing: A survey," *ACM Comput. Surv.*, vol. 42, no. 4, Jun. 2010. [Online]. Available: https://doi.org/10.1145/1749603.1749604

[12] K. Bondalapati and V. K. Prasanna, "Mapping loops onto reconfigurable architectures," in *Field-Programmable Logic and Applications From FPGAs to Computing Paradigm*, R. W. Hartenstein and A. Keevallik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 268–277.

[13] K. Bondalapati, "Parallelizing dsp nested loops on reconfigurable architectures using data context switching," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001, pp. 273–276.

[14] Jong-eun Lee, Kiyoung Choi, and N. D. Dutt, "Compilation approach for coarse-grained reconfigurable architectures," *IEEE Design Test of Computers*, vol. 20, no. 1, pp. 26–33, 2003.

[15] Y. Guo, J. Wang, J. Zhang, and G. Luo, "Formulating data-arrival synchronizers in integer linear programming for cgra mapping," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 943–948.

[16] J. Lee and T. E. Carlson, "Ultra-fast cgra scheduling to enable run time, programmable cgras," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1207–1212.

[17] Y. Miyasaka, M. Fujita, A. Mishchenko, and J. Wawrzynek, "Sat-based mapping of data-flow graphs onto coarse-grained reconfigurable arrays," in *VLSI-SoC: Design Trends*, A. Calimera, P.-E. Gaillardon, K. Korgaonkar, S. Kvatinsky, and R. Reis, Eds. Cham: Springer International Publishing, 2021, pp. 113–131.

[18] M. Wijtvliet, H. Corporaal, and A. Kumar, *Architectural Model*. Cham: Springer International Publishing, 2022, pp. 151–180. [Online]. Available: https://doi.org/10.1007/978-3-030-79774-4_6

[19] T. Kojima, N. A. V. Doan, and H. Amano, "Genmap: A genetic algorithmic approach for optimizing spatial mapping of coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2383–2396, 2020.

[20] B. De Sutter, P. Coene, T. Vander Aa, and B. Mei, "Placement-and-routing-based register allocation for coarse-grained reconfigurable arrays," *SIGPLAN Not.*, vol. 43, no. 7, pp. 151–160, Jun. 2008.

[21] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "Piperench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[22] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "DRESC: a retargetable compiler for coarse-grained reconfigurable architectures," in *FPT*, Dec. 2002, pp. 166–173.

[23] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, "A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 11, pp. 1565–1578, 2009.

[24] S. Das, T. Peyret, K. J. M. Martin, G. Corre, M. Thevenin, and P. Coussy, "A scalable design approach to efficiently map applications on cgras," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 655–660.

[25] S. Dave, M. Balasubramanian, and A. Shrivastava, "Ureca: Unified register file for cgras," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1081–1086.

[26] D. Wijerathne, Z. Li, A. Pathania, T. Mitra, and L. Thiele, "Himap: Fast and scalable high-quality mapping on cgra via hierarchical abstraction," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1192–1197.

[27] L. Chen and T. Mitra, "Graph minor approach for application mapping on cgras," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 3, sep 2014. [Online]. Available: https://doi.org/10.1145/2655242

[28] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: using epimorphism to map applications on CGRAs," in *DAC*. ACM, 2012, pp. 1284–1291.

[29] B. De Sutter, P. Coene, T. Vander Aa, and B. Mei, "Placement-and-routing-based register allocation for coarse-grained reconfigurable arrays," in *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 151–160. [Online]. Available: https://doi.org/10.1145/1375657.1375678

[30] A. Hatanaka and N. Bagherzadeh, "A modulo scheduling algorithm for a coarse-grain reconfigurable array template," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–8.

[31] Z. Li, D. Wijerathne, X. Chen, A. Pathania, and T. Mitra, "Chordmap: Automated mapping of streaming applications onto cgra," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[32] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 268–281.

[33] G. Gobieski, A. O. Atli, K. Mai, B. Lucia, and N. Beckmann, "Snafu: An ultra-low-power, energy-minimal cgra-generation framework and architecture," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1027–1040.

[34] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to cgra mapping," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3195986

[35] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, "A general constraint-centric scheduling framework for spatial architectures," *SIGPLAN Not.*, vol. 48, no. 6, p. 495–506, jun 2013. [Online]. Available: https://doi.org/10.1145/2499370.2462163

[36] Z. Zhao, W. Sheng, Q. Wang, W. Yin, P. Ye, J. Li, and Z. Mao, "Towards higher performance and robust compilation for cgra modulo scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2201–2219, 2020.

[37] 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 *PACT*. ACM, 2008.

[38] S. Dave, M. Balasubramanian, and A. Shrivastava, "Ramp: Resource-aware mapping for cgras," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196101

[39] J. Gu, S. Yin, L. liu, and S. Wei, "Stress-aware loops mapping on cgras with dynamic multi-map reconfiguration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2105–2120, 2018.

[40] M. Canesche, M. Menezes, W. Carvalho, F. S. 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*, vol. 40, no. 8, pp. 1600–1612, 2021.

[41] J. A. Brenner, J. Van Der Veen, S. P. Fekete, J. Oliveira Filho, and W. Rosenstiel, "Optimal Simultaneous Scheduling, Binding and Routing for Processor-Like Reconfigurable Architectures," in *FPL*, Aug. 2006, pp. 1–6.

[42] M. Karunaratne, C. Tan, A. Kulkarni, T. Mitra, and L.-S. Peh, "Dnestmap: Mapping deeply-nested loops on ultra-low power cgras," in *Proceedings of the 55th Annual Design Automation Conference*, ser.

DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196027

[43] 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 *DASIP*, 2010, pp. 168–175.

[44] C. Donovick, M. Mann, C. Barrett, and P. Hanrahan, "Agile smt-based mapping for cgras with restricted routing networks," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2019, pp. 1–8.

[45] S. Yin, D. Liu, L. Liu, S. Wei, and Y. Guo, "Joint affine transformation and loop pipelining for mapping nested loop on CGRAs," in *DATE*, Mar. 2015, pp. 115–120.

[46] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "REGIMap: register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *DAC*. ACM, 2013, pp. 18:1–18:10.

[47] T. Peyret, G. Corre, M. Thevenin, K. J. M. Martin, and P. Coussy, "Efficient application mapping on CGRAs based on backward simultaneous scheduling/binding and dynamic graph transformations," in *ASAP*, Jun. 2014, pp. 169–172.

[48] G. Lee, K. Choi, and N. Dutt, "Mapping Multi-Domain Applications Onto Coarse-Grained Reconfigurable Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 637–650, May 2011.

[49] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, "SPR: An Architecture-adaptive CGRA Mapping Tool," in *FPGA*. ACM, 2009, pp. 191–200.

[50] S. Schulz, O. Bringmann, T. Schweizer, and W. Rosenstiel, "Rotated parallel mapping: A novel approach for mapping data parallel applications on cgras," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, 2014, pp. 1–6.

[51] N. Bansal, S. Gupta, N. Dutt, and A. Nicolau, "Analysis of the performance of coarse-grain reconfigurable architectures with different processing element configurations," in *Workshop on Application Specific Processors, held in conjunction with the International Symposium on Microarchitecture (MICRO), 2003*, 2003.

[52] M. Balasubramanian and A. Shrivastava, "Crimson: Compute-intensive loop acceleration by randomized iterative modulo scheduling and optimized mapping on cgras," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3300–3310, 2020.

[53] S. Mu, Y. Zeng, and B. Wang, "Routability-enhanced scheduling for application mapping on cgras," *IEEE Access*, vol. 9, pp. 92 358–92 366, 2021.

[54] S. Das, K. J. M. Martin, D. Rossi, P. Coussy, and L. Benini, "An energy-efficient integrated programmable array accelerator and compilation flow for near-sensor ultralow power processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1095–1108, 2019.

[55] B. Yuan, J. Zhu, X. Man, Z. Ma, S. Yin, S. Wei, and L. Liu., "Dynamic-ii pipeline: Compiling loops with irregular branches on static-scheduling cgra," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[56] M. L. Anido, A. Paar, and N. Bagherzadeh, "Improving the operation autonomy of simd processing elements by using guarded instructions and pseudo branches," in *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, 2002, pp. 148–155.

[57] K. Chang and K. Choi, "Mapping control intensive kernels onto coarse-grained reconfigurable array architecture," in *2008 International SoC Design Conference*, vol. 01, Nov 2008, pp. I–362–I–365.

[58] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "Branch-aware loop mapping on cgras," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–6. [Online]. Available: https://doi.org/10.1145/2593069.2593100

[59] M. Karunaratne, D. Wijerathne, T. Mitra, and L.-S. Peh, "4d-cgra: Introducing branch dimension to spatio-temporal application mapping on cgras," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[60] S. Das, K. J. M. Martin, P. Coussy, D. Rossi, and L. Benini, "Efficient mapping of CDFG onto coarse-grained reconfigurable array architectures," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 127–132.

[61] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," in *2003 Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 296–301.

[62] M. Balasubramanian, S. Dave, A. Shrivastava, and R. Jeyapaul, "Laser: A hardware/software approach to accelerate complicated loops on cgras," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1069–1074.

[63] C. Sunny, S. Das, K. J. M. Martin, and P. Coussy, "Hardware based loop optimization for cgra architectures," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, S. Derrien, F. Hannig, P. C. Diniz, and D. Chillet, Eds. Cham: Springer International Publishing, 2021, pp. 65–80.

[64] K. Vadivel, M. Wijtvliet, R. Jordans, and H. Corporaal, "Loop overhead reduction techniques for coarse grained reconfigurable architectures," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 14–21.

[65] C. Li, J. Gu, S. Yin, L. Liu, and S. Wei, "Combining memory partitioning and subtask generation for parallel data access on cgras," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 204–209. [Online]. Available: https://doi.org/10.1145/3394885.3431414

[66] Y. Kim, J. Lee, A. Shrivastava, and Y. Paek, "Memory Access Optimization in Compilation for Coarse-grained Reconfigurable Architectures," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 4, pp. 42:1–42:27, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2003695.2003702

[67] Z. Zhao, Y. Liu, W. Sheng, T. Krishna, Q. Wang, and Z. Mao, "Optimizing the data placement and transformation for multi-bank cgra computing system," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1087–1092.

[68] S. Yin, X. Yao, T. Lu, D. Liu, J. Gu, L. Liu, and S. Wei, "Conflict-free loop mapping for coarse-grained reconfigurable architecture with multi-bank memory," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2471–2485, 2017.

[69] S. Jin, W. Seo, Y.-G. Cho, and S. Ryu, "Low-power reconfigurable audio processor for mobile devices," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*, 2014, pp. 369–370.

[70] A. Frumusanu, "The Samsung Exynos 7420 Deep Dive - Inside A Modern 14nm SoC." [Online]. Available: http://www.anandtech.com/show/9330/exynos-7420-deep-dive

[71] A. G, "Architecture apocalypse dream architecture for deep learning inference and compute-versal ai core," in *Embedded World*, 2020.

[72] SambaNova, *Accelerated Computing with a Reconfigurable Dataflow Architecture*, SambaNova systems, 06 2021. [Online]. Available: https://sambanova.ai/wp-content/uploads/2021/06/SambaNova_RDA_Whitepaper_English.pdf

[73] Y. Zhang, N. Zhang, T. Zhao, M. Vilim, M. Shahbaz, and K. Olukotun, "Sara: Scaling a reconfigurable dataflow accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1041–1054.

[74] 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*, vol. 38, no. 12, pp. 2271–2283, 2019.

[75] J. Anderson, R. Beidas, V. Chacko, H. Hsiao, X. Ling, O. Ragheb, X. Wang, and T. Yu, "Cgra-me: An open-source framework for cgra architecture and cad research : (invited paper)," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 156–162.

[76] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "Aurora: Automated refinement of coarse-grained reconfigurable accelerators," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1388–1393.

[77] A. Podobas, K. Sano, and S. Matsuoka, "A template-based framework for exploring coarse-grained reconfigurable architectures," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 1–8.

[78] C. Nicol, "A coarse grain reconfigurable array (cgra) for statically scheduled data flow computing," *Wave Computing White Paper*, 2017.