Slack-aware Scheduling on Coarse Grained Reconfigurable Arrays

Giovanni Ansaloni
and Laura Pozzi
Faculty of Informatics
University of Lugano
via Buffi 13
Lugano, Switzerland
Email: {giovanni.ansaloni},{laura.pozzi}@usi.ch

Kazuyuki Tanimura
and Nikil Dutt
Donald Bren School of
Information and Computer Science
University of California, Irvine
6210 Donald Bren Hall
Irvine, California USA
Email:{ktanimur},{dutt}@uci.edu

***Abstract*—Coarse Grained Reconfigurable Arrays (CGRAs) are a promising class of architectures conjugating flexibility and efficiency. Devising effective methodologies to map applications onto CGRAs is a challenging task, due to their parallel execution paradigm and sparse interconnection topology. In this paper we present a scheduling framework that is able to efficiently map operations on CGRA architectures. It leverages differences in delays of various operations, which a reconfigurable architecture always exhibits at run-time, to effectively route data. We call this ability "slack-awareness". Experimental evidence showcases the benefit of slack-aware scheduling in a coarse-grained reconfigurable environment, as more complex applications can be mapped for a given mesh size and more efficient schedules can be achieved, compared to the state of the art methods.**

I. Introduction

Coarse Grained Reconfigurable Arrays (CGRAs) differ from traditional FPGAs, as they contain coarser basic elements, usually comprising one or more ALUs. In this way, CGRAs trade off bit-level flexibility for improved efficiency, so that they can be deployed as reconfigurable accelerators or reconfigurable functional units to execute performance critical applications.

Application mapping on CGRAs is a complex task, and many strategies have been proposed to accomplish it. However, all previous efforts consider time in discrete chunks, assuming that each operation executed on a CGRA tile consumes a full clock cycle. Our contribution to this research field focuses instead on exploiting *slack*, the difference between the clock period and the critical path of execution of an operation, to combinatorially chain computation and routing.

Slack exploitation has been studied in high-level synthesis [1] and is commonly featured in FPGA placing and routing, but has been neglected by State-of-the-Art CGRA schedulers.

The intuition behind our approach is presented in Figure 1: if communication between cells must be registered, operation B must be executed three cycles after operation A; on the other hand, if cycle time allows it, *and* unregistered communication is supported, B can be executed immediately after A. This strategy presents no penalty in maximum clock frequency if operation A, and routing data through cells, is fast enough compared to the slowest operation performed on the mesh.

978-3-9810801-7-9/DATE11/©2011 EDAA

In this paper we present a novel scheduling strategy that considers both registered and unregistered communication among tiles, resulting in an efficient utilization of computational resources, thus allowing the mapping of more complex kernels, and with a better execution performance, than is done by the state of the art, slack-oblivious methodologies.

We evaluate the present mapping algorithm on the Expression Grained Reconfigurable Array [2] [3], an architectural template of which widely different instances can be derived parametrically, comprising heterogeneous cells and without restrictions on their arrangement.

II. Related Work

Application scheduling on CGRAs poses a novel challenge, mainly due to their sparse interconnection topology lacking a centralized register file.

Some research efforts, taking inspiration from FPGA placing and routing, have considered spatial mapping as a way to maximize execution parallelism. Examples of this strategy are SPKM [4] and SMP [5]. These works acknowledge the dual function (computation and data routing) of CGRA cells but, as opposed to us, neglect the opportunity to combinatorially chain cells to speed up execution.

A modulo scheduling approach has instead been taken by Mei [6], Hatanaka [7] and Park [8]. In these works, both space and time dimensions are considered during mapping—as opposed to a spatial approach—borrowing from the modulo scheduling technique employed on VLIW architectures [9]. However, [6] – [8] also overlook critical paths issues by assuming only registered connections between tiles, so that each operation consumes an entire clock cycle.

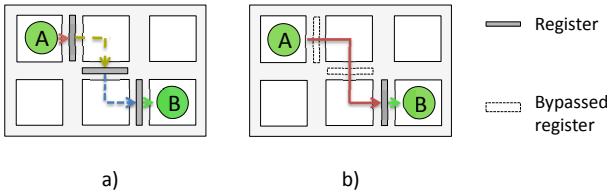

Fig. 1. a) registered and b) unregistered routing through a CGRA mesh.

TABLE I
CRITICAL PATH DELAY OF DIFFERENT EGRA OPERATIONS.

	route	RAC						mult	mem
		bool-bool	bool-sh	bool-add	sh-sh	sh-add	add-add		
critical path (ns)	0.31	0.67	0.85	0.98	1.03	1.16	1.29	1.37	0.85
% of a 1.37 ns **clock period**	23	49	62	71	75	84	94	100	62
routing hops	3	2	1	1	1	0	0	0	1

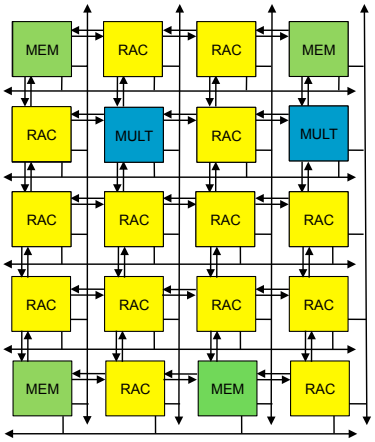

Fig. 2. Example EGRA instance composed of 2 multipliers, 4 memory cells and 14 RACs (Reconfigurable ALU Clusters).

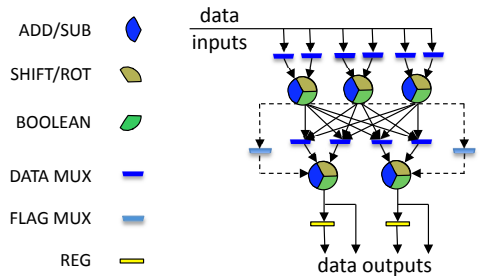

Fig. 3. Structure of a 3-2 Reconfigurable ALU Cluster.

III. TARGET ARCHITECTURE

An EGRA instance is composed of a mesh of cells of parametrically determined size, communicating with nearest neighbour connections and local horizontal and vertical buses, as described in Figure 2. Tiles at each location of a mesh can be one of three basic types: multiplier, cluster of ALUs (Reconfigurable ALU Cluster, RAC) or memory, as decided by design parameters dictated by a *machine description*.

These three types of tiles can accommodate the most used operations present in computational kernels of embedded systems' applications; nonetheless, different types of cells can be integrated in the template as needed by implementing their internal structures.

Each tile type can be customized at design time to fit intended target applications. As an example, memory cells can be instantiated as single or dual-port, with addressing modes ranging from 8 to 32 bits. Multipliers can support signed, unsigned or both types of multiplication.

Structure of RAC tiles.. RACs, depicted Figure 3, are composed of rows of ALUs connected through switch-boxes and supporting various operations, including if-conversion through the usage of 1-bit *flags*. The number of rows in RAC, the number of ALUs in each row and the operations supported by ALUs are again machine description parameters; indeed, [2] presents a design space exploration on RAC structure, from which emerges that two-rows RACs, like the one illustrated in Figure 3, achieve a good overall performance.

Delay of tiles. Synthesis data shown in Table I, first row, highlights the different critical path of array tiles, depending on their type, and (in the case of RAC tiles), on the operation that they are configured to perform. To extract these results, we considered RACs composed of five ALUs in two rows (corresponding to the scheme in Figure 3), a multiplier capable of both signed and unsigned multiplication and single-ported 1kB memory cells with 32-bits data addressing. We employed Design Compiler from Synopsys and TSMC 90nm libraries.

Assuming a clock period equal to the critical path of the slowest cell (the multiplier), Table I shows, in the second row, the percentage of clock period taken by each cell performing their supported operations. The third row of Table I shows how many routing hops can be performed *after* computation and in the same clock cycle, without violating timing constraints. For example, a RAC configured to execute two boolean operations can chain two routing hops before exhausting cycle time, while a memory operation just one.

This data highlights how heterogenous computation times can be leveraged to increase schedulability of kernels without increasing clock period.

IV. SCHEDULING FRAMEWORK

Goal of the scheduler is to modulo map a Data Flow Graph (DFG), representing an iteration of a computational kernel, onto the architecture, i.e., onto a scheduling space representing a computational mesh.

In a nutshell, the proposed scheduling algorithm starts from an initial mapping that is high performance but possibly invalid, and iterates in search of a valid solution via a simulated annealing strategy. If a valid solution is not found with the currently sought high performance, the performance is lowered and the iteration starts again. Each step of the proposed algorithm will now be explained.

Expansion of the input DFG. To account for the dual use of CGRA cells (computation and routing), the input DFG is expanded by inserting routing nodes. On each edge, the number of routing nodes must be sufficient to completely traverse the scheduling space, whose time dimension is bounded by the maximum as-late-as-possible (ALAP) among operations to be mapped [9], while its space dimension corresponds to the physical size of the reconfigurable mesh. In the case of

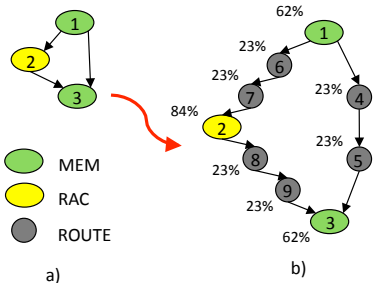

Fig. 4. Routing nodes insertion on a DFG, with the annotation of the critical path length relative to the clock period.

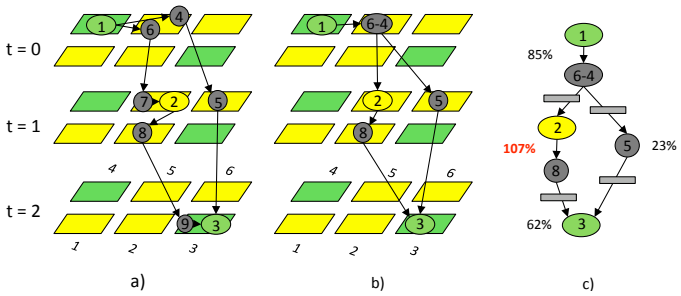

Fig. 5. a) Expanded DFG mapping on the scheduling space. b) After redundant nodes deletion. c) Resulting DFG with annotation of routing times. The combinatorial chain of nodes 2 and 8 violates the timing constraint.

operation node	1	2	3
1	-	0	0
2	-	-	1
3	-	-	-

a)

	cell index					
	1	2	3	4	5	6
occupancy	0	1	1	1	2	1

b)

Fig. 6. a) Slack Violation Table and b) Modulo Resource Table derived from the scheduling space in Figure 5.

the considered example, this amounts to 2 routing nodes, and the graph is expanded accordingly (Figure 4).

Generation of an initial schedule. The scheduling space is a three dimensional graph replicating the CGRA structure size, cells' types and topology $max(ALAP(op))$ times (3 times in the example). The graph edges connect to both the same time plane (representing unregistered connections) and to the following plane (representing registered connections).

To generate an initial schedule, three steps are performed (and illustrated in Figure 5a-b): first, operations are placed in the scheduling space on cells that support them and respecting their precedence constraints. Then, routing nodes are mapped to connect such operations, by employing the A* algorithm [10]. Finally, redundant routing nodes are deleted; routing nodes can be redundant either because they carry the same data of a node already scheduled on the same position (node 4 or 6, Figure 5a-b) or if they are placed at the position of their successor operation node (node 7 and node 9, Figure 5a-b).

Figure 5c shows the mapped DFG, decorated with registers among planes, and annotated with delays.

Calculating the cost of a schedule. A tentative schedule can be invalid due to either timing violations or resource overuse.

Timing violation occurs when a path from register to register exceeds cycle time. Delays over paths are calculated, and a table is kept that indicates the amount of violation on each edge. In the example, see Figure 5c and 6a, the Slack Violation Table (SVT) indicates a violation between nodes 2 and 3, as node 2 is computed at t = 1, and its output is routed to the cell below it without registering the result.

Resource overuse occurs when more than what can be supported by a cell is mapped onto it. This can happen in two cases: 1) when a cell is being used to route more than a single value, 2) when a cell is being used to compute an operation, *and* to route a different value.

Information on resource overuse is stored in the Modulo Resource Table (MRT). Modulo scheduling aims at maximizing parallelism by pipelining successive iterations of kernels execution; the distance (in clock cycles) between two iterations is defined as the *Initiation Interval* (II). To account for pipelining, the scheduling space must be folded according to the II when considering resource overuse; the resulting MRT is composed of exactly II rows, and contains the usage of each resource added modulo II. Figure 6b illustrates the Modulo Resource Table for the initial placement in Figure 5 considering II = 1. It can be noticed that cell 5 is overused, as it hosts node 6-4 at t = 0 and node 2 at t = 1.

This scheme can be easily extended to more complex topologies, including shared communication links, modeled as resources able to accommodate routing cells only. Indeed the results presented in Section V consider local buses. Once the MRT and the SVT are computed, a placement cost can be derived by adding up overuse and timing violations.

Iterating in search of a valid solution. If the current schedule is not valid, a new one is created: an operation node is unscheduled together with its successor and predecessor routing nodes, freeing up related resources; the operation node is then remapped and related routing is performed to and from the node; a new cost value is computed and the move is accepted depending on its cost and the current (ever-decreasing) $temperature$. The process is repeated until a valid mapping is found (with $placement\_cost = 0$) or if the maximum number of tries has been reached.

Lowering Performance. If a valid solution has not been found after a number of iterations, a less aggressive mapping, of lower performance, is tried. This can be obtained by either increasing nodes mobility by augmenting their ALAP, or by increasing the Initiation Interval. The former can be beneficial to overcome timing violations, the latter to alleviate resource overuse.

V. Experimental Results

We evaluated the benefit of slack-awareness to perform modulo scheduling of DFGs to CGRAs. To do so, we compared a setting where operations consume an entire clock cycle (slack-oblivious scheduler, as employed by Mei [6], Hatanaka [7] and Park [8]) to one in which slack-awareness is applied to allow for combinatorial chaining of operations.

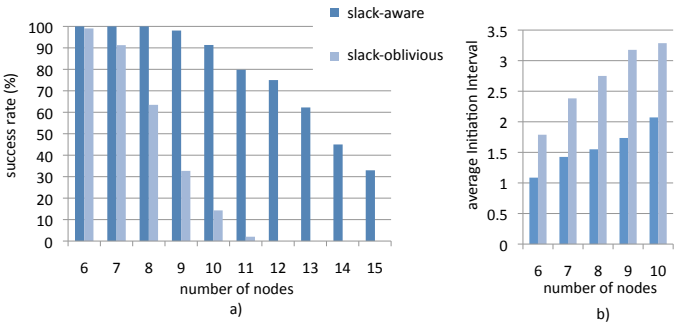

Fig. 7. Slack-aware vs. slack-oblivious modulo scheduling: success rate of test DFGs (a) and resulting Initiation Interval (b).

Experimental settings. We scheduled automatically generated DFGs on the EGRA instance presented in Figure 2, using the two above-mentioned methodologies. DFGs ranged from 6 to 15 operation nodes, and we investigated one hundred of them for each DFG size.

We considered DFGs with diverse shapes and characteristics: nodes were set to have one or two predecessors, with 50% probability in each case; nodes' types were randomly assigned with a probability matching the composition of the target mesh (10% multiplications, 20% memory operations, 70% RAC operations). The clock period was set to be equal to 1.37ns (the time used by the slowest operation, multiplication); consequently, allowed unregistered routing for the slack-aware scheduler followed data presented in Table I. Three thousand simulated annealing cycles were performed before increasing the Initiation Interval; application mapping failed when II reached $max(ALAP)$, a situation where loops are not pipelined at all.

Slack-aware scheduling maps more DFGs. Data plotted in Figure 7a shows the percentage of successful mappings for each DFG size using slack-aware and slack-oblivious mappings. The graph highlights the efficiency of slack awareness as, for example, 90% percent of 10-nodes DFGs where successfully mapped using slack-aware modulo scheduling, while the same figure is around 10% for a slack-oblivious strategy.

Slack-aware scheduling achieves better mappings. In addition to being able to map more DFGs, slack awareness also improves performance of mapped applications. Figure 7b compares the average Initiation Interval of DFGs which were successfully mapped with a slack-aware or a slack-oblivious strategy. The slack-aware scheduler achieves on average a 33% smaller II, corresponding to a 33% faster execution of the mapped kernel.

Real benchmarks. We considered DFGs of computational kernels extracted from the EEMBC [11] benchmark suite. We again employed the EGRA instance described in Figure 2 and we retained the simulated annealing parameters used for the previous experiments; we mapped each kernel one hundred times with different initial conditions. Table II illustrates the size of the DFGs before and after graph expansion, the percentage of successful mappings and the average Initiation Interval achieved with a slack-aware method compared to a slack-oblivious one.

TABLE II
SCHEDULABILITY AND PERFORMANCE OF BENCHMARK DFG KERNELS SCHEDULED USING DIFFERENT METHODS.

B.mark	DFG nodes	Exp. DFG nodes	Scheduling method	Success (%)	II
autocorr	5	35	slack aware slack oblivious	100 100	1.04 1.59
conven	5	35	slack aware slack oblivious	100 92	1.00 2.32
aifirf	6	48	slack aware slack oblivious	100 51	2.00 2.25
mpegcorr	8	62	slack aware slack oblivious	100 0	2.66 -
iquant	9	69	slack aware slack oblivious	100 0	2.08 -
fbital	9	81	slack aware slack oblivious	100 0	3.22 -

Results are in line with the ones obtained for randomly generated DFGs: slack-aware modulo scheduling is able to map all six benchmarks, while a slack-oblivious strategy fails in the three most complex kernels; when both succeed, slack-aware scheduling results in a more performing mapping (with a smaller II).

REFERENCES

[1] M. Sivaraman and S. Aditya, "Cycle-time aware architecture synthesis of custom hardware accelerators," in *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, Oct. 2002, pp. 35–42.

[2] G. Ansaloni, P. Bonzini, and L. Pozzi, "Design and architectural exploration of expression-grained reconfigurable arrays," in *Proceedings of the 6th Symposium on Application Specific Processors*, Anaheim, CA, Jun. 2008, pp. 26–33.

[3] G. Ansaloni, P. Bonzini, and L. Pozzi, "Heterogeneous coarse-grained processing elements: a template architecture for embedded processing acceleration," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Nice, France, Apr. 2009, pp. 542–547.

[4] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, R. Jeyapaul, and Y. Paek, "SPKM: A novel graph-drawing based algorithm for application mapping onto coarse-grained reconfigurable architectures," in *Proceedings of the Asia and South Pacific Design Automation Conference*, Seoul, South Korea, Jan. 2008.

[5] M. Ahn, J. Yoon, Y. Paek, Y. Kim, M. Kiemb, and K. Choi, "A spatial mapping algorithm for heterogeneous coarse-grained reconfigurable architectures," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. European Design and Automation Association 3001 Leuven, Belgium, Mar. 2006, pp. 363–368.

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

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

[8] H. Park, K. Fan, M. Kudlur, and S. Mahlke, "Modulo graph embedding: Mapping applications onto coarse-grained reconfigurable architectures," in *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, Oct. 2006, pp. 136–146.

[9] B. Ramakrishna Rau, "Iterative Modulo Scheduling," *International Journal of Parallel Processing*, vol. 24, no. 1, pp. 2–64, Feb. 1996.

[10] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[11] "EEMBC website. *http://www.eembc.org/*."