2018 International Conference on Field-Programmable Technology (FPT)

Compact Area and Performance Modelling for CGRA Architecture Evaluation

Kuang-Ping Niu and Jason H. Anderson
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, ON M5S3G4 CANADA
kuangping.niu@mail.utoronto.ca, janders@eecg.toronto.edu

***Abstract*—We present area and performance models for use in coarse-grained reconfigurable array (CGRAs) architectural exploration. The area and performance models can be computed rapidly and are incorporated into the open-source *CGRA-ME* architecture evaluation framework [1]. Area is modelled by synthesizing (into standard cells) commonly occurring CGRA primitives in isolation, and then aggregating the component-wise areas. For performance, we incorporate a fully fledged static-timing analysis (STA) framework into CGRA-ME. The delays in the STA timing graph are annotated based on: 1) a library component-wise delays for logic/memory, and 2) a fanout-based delay estimation model for interconnect. Performance and area are modelled for both performance-optimized and area-optimized standard-cell CGRA implementations. Accuracy of the area and performance models is within 7% and 10%, respectively, of a fully laid-out standard-cell CGRA implementation.**

I. Introduction

Coarse-grained reconfigurable arrays (CGRAs) are programmable logic devices with bus-based interconnect and large coarse-grained ALU-like logic blocks. This is in contrast to field-programmable gate arrays (FPGAs), wherein individual logic signals are routed independently, and the core logic element is a look-up-table – essentially a hardware realization of a truth table. As such, a CGRA is "less programmable" than an FPGA, with two key implications: 1) there is less area overhead associated with programmability (good) since they are programmable by bus instead of bit, and 2) CGRAs are less flexible than FPGAs (good and bad), for the same reason. The former implication brings CGRAs closer to custom ASICs on the spectrum of power, speed, and silicon area cost. The latter implication implies, firstly, that CGRAs are "easier" to map to, as the reduced flexibility means that CAD tools are responsible for fewer decisions, and secondly, that CGRA architectures must be aligned with application requirements if they are to be useful.

While a plethora of CGRA architectures have appeared in the literature (e.g. [2], [3]), and a few commercial CGRAs have been developed [4], [5], they are less studied than alternative computing platforms, such as FPGAs, CPUs and GPUs. For such platforms, architecture modelling and evaluation frameworks exist, allowing hypothetical architectures to be targeted, tested, and compared with baseline architectures. For CGRAs, the only such publicly accessible framework is CGRA-ME [1], [6], which is under active development at the University of Toronto. In this paper, we extend CGRA-ME to allow accurate high-level modelling of CGRA area and performance, without the requirement for full detailed physical implementation.

Architecture modelling frameworks for FPGAs, CPUs, and GPUs allow hypothetical architectures to be modelled and evaluated at an abstract level, where estimation models are applied to gauge area, performance, and power. For example, in Verilog-to-Routing (VTR) [7], area is estimated by counting the number of minimum-width transistors required by the modelled FPGA. The advantage of this high-level approach is to facilitate the rapid exploration of the architectural space. Once good points in the architecture space have been identified, a more detailed implementation of the desirable architectures can be performed to refine early estimates. If, on the other hand, a full standard-cell or custom VLSI implementation were performed for each architectural candidate, the breadth of exploration would be severely hindered by lengthy ASIC CAD tool runtimes – likely hours or days to produce each datapoint.

The CGRA-ME framework [6] accepts a CGRA architecture description as input (specified in an XML-based language or using a `C++` API), as well as an application benchmark to be mapped onto the architecture. CGRA-ME is capable of mapping benchmarks onto an in-memory model of the architecture, as well as producing Verilog RTL for the architecture, and a configuration bitstream to configure the architecture to realize the application benchmark's functionality. While a previous work [6] demonstrated that it was possible to push the Verilog RTL through commercial ASIC CAD tools, CGRA-ME offered no capability for high-level area, performance and power modelling. The work described in this paper overcomes this limitation for performance and area metrics.

Our approach to area and performance modelling is based on the notion that CGRAs are composed of commonly occurring primitives, including multiplexers of various sizes, functional units with specific arithmetic capability, registers, register files, I/Os, and so on. As such, we use standard-cell ASIC tools to create models of area and delay for each such primitive, thereby producing a characterization library. Several such libraries are constructed, representing, for example, area-optimized or delay-optimized implementations of the primitives by the ASIC tools. The two optimization targets are selected because die size and performance are common design objectives in varied

978-1-7281-0214-6/18/$31.00 ©2018 IEEE
DOI 10.1109/FPT.2018.00028

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.

applications. Overall CGRA area can then be estimated by aggregating primitive component areas.

For performance, the individual primitive timing information is insufficient to gauge the speed performance of an application as implemented on a modelled CGRA. We therefore incorporated full static timing analysis (STA) into CGRA-ME, leveraging an open-source STA framework, also used in the VTR. The primitive delays are annotated onto the framework's timing graph, allowing a critical path delay report to be produced (in addition to a variety of other reports). In an experimental study, we compare the area and performance of the primitive-based model to that of a full ASIC implementation of the modelled CGRA, and demonstrate that the rapid estimation model produces accurate estimates. The primary contribution of this work is the capability to perform fast and accurate area/performance estimation for any given architecture modelled within CGRA-ME.

II. Related Work

Aside from CGRA-ME, there are few generic CGRA modelling and exploration frameworks. In [8], a commercial framework was extended to support high-level modelling of CGRAs. SPR [9] is another generic CGRA mapping tool. However, these previous works present performance results measured in cycle-count latency instead of the critical path delay on a specified architecture, and they did not analyze area usage. Suh et al. [10] described architectural design-space exploration (DSE) for the Samsung Reconfigurable Processor (SRP), which improved SRP's performance, and chip area. Kim et al. [11] demonstrated DSE on a reconfigurable array architecture (RAA). Both DSE studies provide performance, area, and power results. Yan et al. [12] described performance and area estimation for a CGRA implemented as an FPGA overlay. Although the above studies include area and performance results, they are either tied to one specific CGRA architecture or implementation technology.

Bansal et al. [13] provided a detailed study on how different configurations of processing elements could affect performance measured in cycle-count latency, but the paper does not elaborate on critical path delay since they do not specify the implementation technology. Ohm et al. [14] described an accurate and very fine-grained estimation technique for high-level design. Each basic building block is realized in various implementations with standard-cell technology, and performance results of all implementations are later used in a top-level estimator to select implementations for all instances of basic building blocks used in a high-level design, while satisfying all specified design requirements. The tool then reports an estimate of total area. The CGRA-ME framework offers a generic mapper based on user-specified architecture, and this work extends the framework to estimate performance and area by accepting a user-specified area/performance profile based on a user-selected implementation technology.

III. CGRA-ME Framework

CGRA Modelling and Exploration (*CGRA-ME*), as the name suggests, is a tool which offers architecture exploration for CGRAs, as well as permitting research on CGRA CAD algorithms. With CGRA-ME, both the architecture specification, as well as an application benchmark, are input to the toolflow. CGRA-ME permits the scientific evaluation and comparison of hypothetical CGRA architectures.

The CAD flow depicted in Fig. 1 starts from the top to bottom. The main user inputs are the architecture description, and an application benchmark. Through the LLVM kernel-extraction pass, key computations of the application benchmark are extracted and are represented as dataflow graphs (DFGs). The architecture interpreter accepts an architecture specification (either in an XML language or using a `C++` API) as input and builds an in-memory device model of the architecture. The in-memory model is a Modulo Routing Resource Graph (MRRG) [15], which has been proposed in prior work on CGRAs. In the MRRG, all functional units and routing resources are represented as nodes, and electrical connections between these elements are represented as edges. The in-memory architecture model can also be used to generate a Verilog implementation of the architecture.

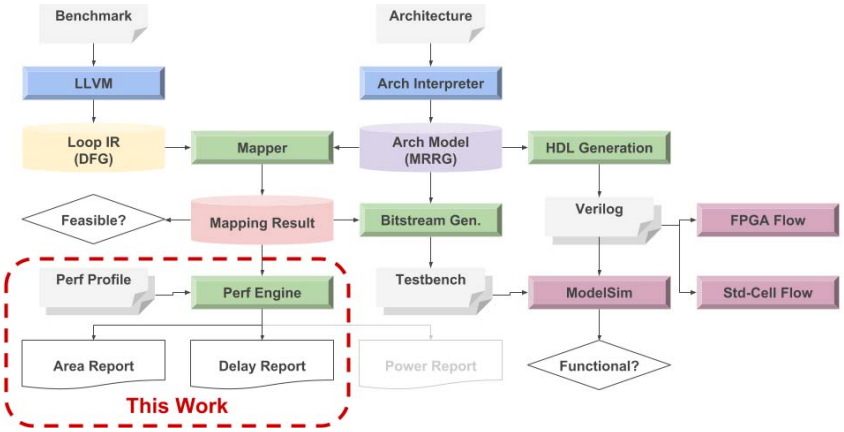


Fig. 1: CGRA-ME framework overview

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.

The mapping step will schedule, place, and route the DFG onto the MRRG. That is, in mapping, each computation in the DFG must be associated with a functional unit node in the MRRG, and each connection between nodes in the DFG (data dependency) must be mapped to a path of nodes/edges within the MRRG, thereby connecting the relevant functional unit nodes, accordingly. The mapper offers two choices for algorithm – a simulated-annealing-based approach [1], and an integer linear programming (ILP)-based approach [16]. If the mapping is feasible, this implies the modelled CGRA can be configured to realize the computations of the DFG. The mapping result and architecture model can then be used to generate a bitstream to configure the CGRA, and verify functionality through RTL simulation.

Performance and area estimation following the mapping step are realized. The inputs to the estimation engine are: 1) the application benchmark as mapped onto the modelled CGRA, and 2) profiles of the area and delay of commonly used CGRA primitives. The estimation engine and characterization data are elaborated upon in the next section.

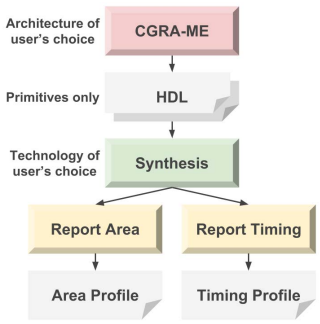


Fig. 2: General steps to retrieve primitive module characteristics.

IV. CGRA-ME Estimation Engine

A. Characterization of Primitive Modules

The area and performance estimation engine depends on the characterization of *primitive modules* that are common to most CGRAs, such as multiplexers, functional units, register files and so on. Fig. 2 shows the steps to acquire the characterization. At a high level, our approach is to implement such primitive modules, one-by-one, in a target standard-cell library, and store the area and delay results for each in a database. The database entries then serve as inputs to the area/performance estimation engine. The approach is analogous to that used for commercial FPGAs, wherein delay characterization data for key sub-circuits is stored in a database, whose entries are then recalled during static-timing analysis (STA).

For the purposes of this work, primitive modules comprise:

- Logical and arithmetic functional operators, including add, subtract, multiply, XOR, AND, shift, etc.
- Multiplexers with various numbers of inputs.
- Register files with various numbers of input/output ports.
- Registers.
- Tri-state buffers.

where the operands to the above are 32-bits wide (in this case study, however, arbitrary width is possible).

In an ASIC standard-cell flow, constraints can be applied during technology mapping to optimize for area, delay, or a combination of the two. Individual cells (e.g. 2-input NAND) are available in multiple drive strengths, and delay-driven mappings will tend to use larger cells with higher drive strengths. CGRAs may be used in a high-performance context, or a low-power embedded context, as and such, we opted to generate two databases of area/performance values for the primitives: an *area-optimized* database wherein the ASIC tools were executed with a minimum-area objective, and a *delay-optimized* database wherein the ASIC tools were executed with a minimum-delay objective. This permits a human architect

	Area [μm^2]		Critical Path Delay [ns]	
Target	Area Optimized	Delay Optimized	Area Optimized	Delay Optimized
op_add_32b	168	536	2.78	0.37
op_sub_32b	190	539	2.80	0.40
op_multiply_32b	2860	3008	1.12	1.10
op_and_32b	43	43	0.03	0.03
op_or_32b	43	74	0.05	0.04
op_xor_32b	64	64	0.06	0.06
op_shl_32b	456	491	0.53	0.43
op_ashr_32b	456	470	0.53	0.45
op_lshr_32b	456	470	0.53	0.45
mux_2to1_32b	74	88	0.06	0.07
mux_4to1_32b	147	166	0.06	0.06
mux_5to1_32b	179	283	0.06	0.11
mux_6to1_32b	215	245	0.09	0.07
mux_7to1_32b	252	340	0.07	0.07
mux_8to1_32b	286	325	0.07	0.07
RF_1in_2out_32b	1123	1231	0.07	0.08
RF_4in_8out_32b	9307	11568	0.17	0.20
register_32b	214	222	0.01	0.01
tristate_32b	86	102	0.41	0.22
const_32b	214	222	0.01	0.01

TABLE I: Database of area and critical path delay of the primitive modules as mapped into the NanGate FreePDK45 45*n*m standard-cell library.

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.

user of CGRA-ME to select between the databases according to the intended CGRA usage.

Table I shows a portion of the two databases used in this work. The left-most column lists the primitive. The next two columns give the area of each primitive in each of the two databases. The right-most two columns give the data, in *n*s for each of the primitives. Broadly speaking, we observe that the delay-optimized primitives are larger and faster than the area-optimized primitives.

All area/performance results are based on targeting the 45*n*m NanGate FreePDK45 Generic Open Cell Library [17]. Synopsys Design Compiler is used for standard-cell technology mapping. Cadence Innovus is used for placement and routing. Synopsys PrimeTime is used for standard-cell timing analysis. Note that the estimation engine is independent of the specific standard-cell target technology, because the ASIC design flow is the same regardless of technology node or standard-cell library. This means that users can redefine entries in the database based on the standard-cell library, and IP libraries available to them, and results of our CGRA estimation engine will properly reflect the target technology and IP. The databases are human-readable `INI` files and easy to read/modify.

B. Area Modelling

As mentioned above, the CGRA-ME framework has an architecture interpreter, which constructs an in-memory model of the targeted CGRA. The in-memory model is represented hierarchically, with a top-level module representing the *entire* CGRA, and second-level modules representing CGRA tiles in the two-dimensional array, and so on. Within CGRA-ME, an architect may specify modules with arbitrary levels of hierarchy. However, at the bottom of the module hierarchy lie primitive modules, which precisely align with those discussed in the previous section, for which area and delay data are contained in the database.

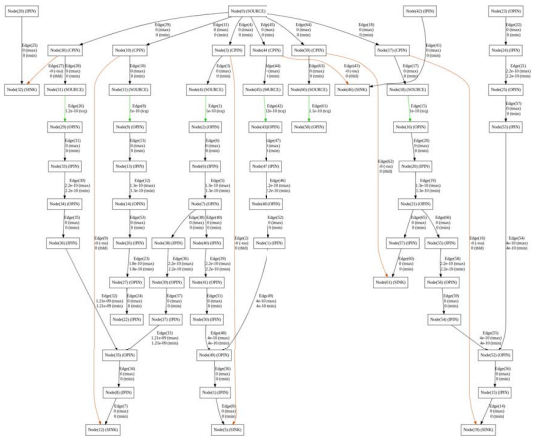
Area estimation therefore comprises "walking" the hierarchical in-memory CGRA model from the top down to the primitives. The area of each primitive module is drawn from the database and aggregated upwards. At the end of the traversal, an estimate of the total CGRA area is available and reported to the user. Likewise, a report also shows the estimated area at each lower level of the hierarchy, giving the architect visibility into the breakdown of area for the modelled CGRA. In the experimental study below, we compare this straightforward primitive-aggregating estimation approach to the actual area after a full ASIC place and route of a CGRA.

C. Delay Modelling and Timing Analysis

As with an FPGA, the critical path of an application benchmark implemented on a CGRA depends on the application's mapping, placement and routing within the CGRA, as well as the circuit delays within the CGRA device (from the characterization database discussed above).

We integrated an open-source static-timing analysis (STA) engine into CGRA-ME. The STA engine, called *Tatum*, is also used within the VTR project [18]. Tatum performs timing

(a) Mapped MRRG



(b) Timing graph

Fig. 3: Mapped MRRG and timing graph of the "sum" benchmark.

analysis using a timing graph, wherein nodes represent pins on electrical components, and edges represent connections between pins. Delays are then annotated onto the edges of the timing graph. Tatum has an easy-to-use `C++` API that allows one to create the timing graph and perform the delay annotation. Following this, Tatum performs timing analysis on the graph, and can generate a critical-path delay report. That is, Tatum includes the functionality for the familiar STA tasks of forward delay propagation to find the worst-case timing paths in a design, and backward propagation of slacks [19] to find the timing slack on each edge of the graph. Tatum can also be extended to accept an `SDC` (Synopsys Design Constraints) file as input, to allow user-control over the timing analysis (e.g. setting of false paths, selection of specific timing to analyze).

To integrate Tatum into CGRA-ME, we use the mapping results of the application benchmark's DFG onto the MRRG (CGRA device module). In essence, we walk the *used* part of the CGRA for the application benchmark and construct the timing graph in Tatum. For example, the used input and output pins on a CGRA multiplexer become nodes in Tatum's timing graph, connected by an edge. The delays on the timing graph's edges are then annotated based on the characterization database

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.

discussed above. Fig. 3 illustrates the *sum* benchmark's mapped MRRG, and the corresponding timing graph in Tatum. Observe that the timing graph is generally larger than the mapped MRRG, because nodes in the timing graph represent pins, whereas nodes in the MRRG are more coarse-grained.

1) Interconnect Delay Estimation

The characterization database contains delays for each type of primitive, however, interconnect delays are a growing contributor to total delay in deep-submicron VLSI technology. Such delays are not accounted for in the database. Moreover, as our aim is to provide a high-level performance estimation capability, we must *estimate* interconnect delay prior to placement and routing (i.e. without detailed knowledge of wirelength, capacitance, resistance). Therefore, to improve the delay-modelling capabilities of CGRA-ME, we constructed a simple fanout-based interconnect delay estimation model. Fanout is widely used for a proxy for wirelength in pre-layout delay estimation [20].

Specifically, for each type of primitive, we attached the outputs of the primitive to various numbers of fanout registers. We then performed a full synthesis, placement, and routing into standard-cells, followed by ASIC STA (using Synopsys PrimeTime). We extracted the delay from the outputs of the primitive to the fanout registers. This allowed us to create a model associating primitive fanout with delay. Fig. 4 plots the relationship between fanout register count and average fanout delay of all primitives, showing a line of best fit. As shown, the interconnect delay is roughly linear with the fanout. Using the MRRG fanout and the linear fanout-delay model, we can optionally annotate estimated interconnect delays onto Tatum's timing graph.

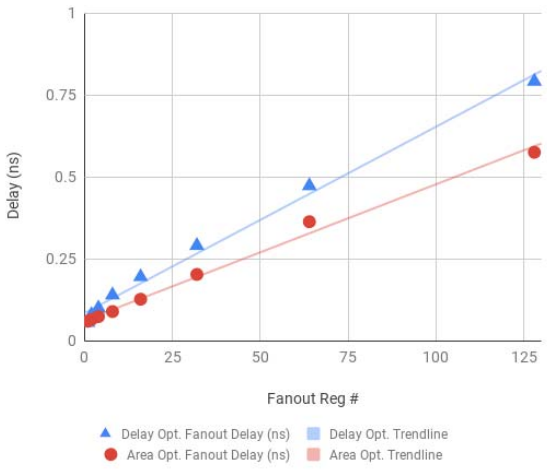


Fig. 4: Averaged fanout-delay of all primitive modules.

V. Experimental Study

The experimental study compares the estimates of area and performance (from the modelling described above), with the accurate area and performance values from a full VLSI CAD flow: synthesis, placement, and routing.

The CGRA-ME framework includes a number of CGRA architecture models, as well as a set of application benchmark circuits. We selected one of the CGRAs (called ADRES, described below), and 8 application benchmarks. The benchmarks used in this study consist of 6 to 31 DFG nodes. We used CGRA-ME to generate RTL for the CGRA, which we then pushed through the full ASIC flow twice: once targeting minimum area, and once targeting minimum delay. The layouts were floorplanned to ensure the physical layout of CGRA tiles matched the logical layout (i.e. to ensure tiles in neighbouring rows/columns were indeed physically adjacent). For placement and routing, design density was constrained to 80% and the aspect ratio to 1.2. Following placement and routing, the CGRA area is precisely known.

For performance, we performed an RC extraction on the layout and executed post-layout timing analysis with Synopsys PrimeTime. However, running PrimeTime on the CGRA netlist does not yield meaningful results, as CGRAs, when unconfigured, contain many combinational loops. As such, for each of the application benchmarks, we also provide a (`.sdc`) constraint file to PrimeTime, directing the tool to *only* analyze timing paths in the *used* part of the CGRA for the particular benchmark's mapping. This allows us to find the critical-path delay, for each benchmark, in the full ASIC CGRA implementation in an accurate manner.

We use the ADRES CGRA architecture [6], [21] as the test vehicle to evaluate the high-level area/performance estimation engine. Fig. 5 provides a high-level view of the architecture. From the top to bottom, the architecture is equipped with a row of I/O ports, a large register file (DRF) shared by the first row of processing elements (PE), a 4×4 grid of PEs, smaller register files (RF) each coupled with one PE excluding the first row, and memory interface ports (MP) each connecting to a row of PEs. Each PE consumes and provides data to the nearest orthogonal neighbor PEs. PEs on edges are also connected by toroidal buses, highlighted in red and blue in Fig. 5, representing vertical and horizontal toroidal connections, respectively.

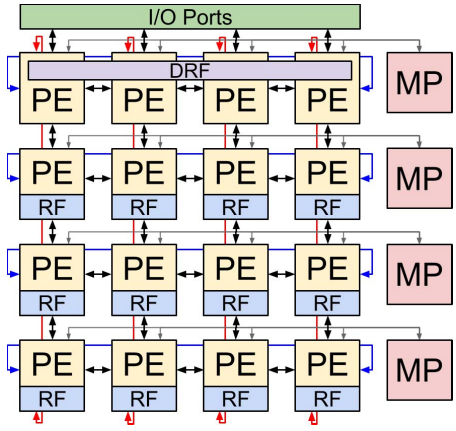


Fig. 5: High-level view of the ADRES-like architecture used in the experimental study.

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.

Fig. 6 shows layouts of ADRES for both the area/performance targets side by side, in the same scale. Table II shows the total chip area of both targets in the row labelled Baseline. Table III summarizes the critical path delays of the benchmarks. Note that the memory interface ports (MP) are not included in the performance/area results, as these are typically proprietary IPs and our focus is on the CGRA aspects of area/performance.

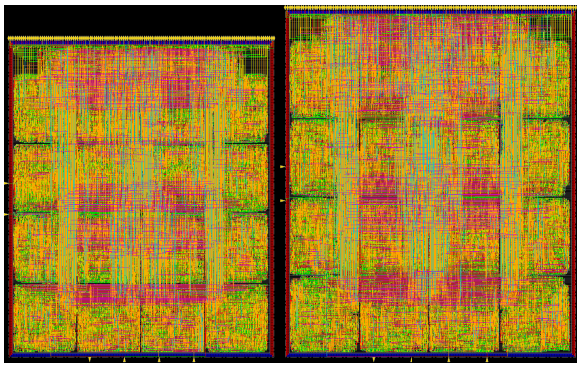


Fig. 6: Standard-cell PnR architecture layout of ADRES – area-optimized (left) vs. delay-optimized (right) side-by-side on the same scale.

	Area-optimized	Delay-optimized
Baseline [μm^2]	125945.3	151203.0
CGRA-ME [μm^2]	117743.0	141068.0
Diff	-6.51%	-6.70%

TABLE II: Total core area of area-optimized and delay-optimized designs, as well as estimation by CGRA-ME.

	Area-optimized [ns]	Delay-optimized [ns]
conv2	4.37	2.77
conv3	4.46	3.20
mac	4.51	2.92
mults1	4.51	2.77
nomem1	4.33	2.72
simple	3.24	2.71
simple2	3.24	2.71
sum	4.13	2.96

TABLE III: Critical path delay of the 8 selected benchmarks for area-optimized and delay-optimized targets from PrimeTime STA.

A. CGRA-ME Estimation Results

We first examine area estimation, where the estimates are shown in the CGRA-ME row of Table II. Comparing the two rows of the table, we see close alignment between the estimates and actual area values for the entire CGRA. The estimates are about 6% lower than the actual layout values. The results confirm that aggregating primitive module areas provides a good estimate of overall CGRA area. The 6% gap between our tool and the actual areas is anticipated, since our tool does not account for the area contributed by configuration cells. We do not take configuration cells into account in our estimates, because their detailed implementation may vary depending on the CGRA (e.g. SRAM cells or flip-flops). However, the number of configuration bits can be estimated to further improve the result. For the baseline results, configuration cells are implemented as flip-flops, connected in a scan chain.

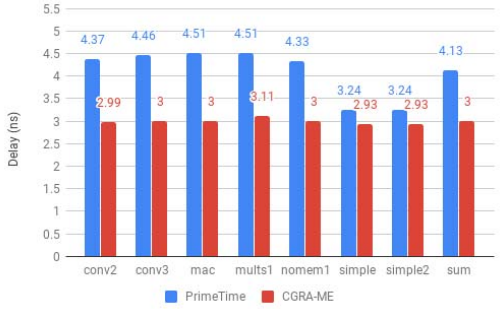
Turning now to the performance results, Figs. 7a and 7b show the critical path delays of the 8 benchmarks. The blue bars show the actual post-routing critical path delays; the red bars represent the estimates provided by our model. In this case, the estimates do not include any of the fanout-based interconnect delay estimation discussed above. Observe that the estimated critical path delays are almost always optimistic (i.e. the estimated delays are smaller than the actual delays). The average error in the critical-path delay estimation is 1.25ns, 43%. Out of the 16 benchmarks (in the two figures), the critical path reported by the estimator was the same as the actual critical path in 7 of the benchmarks. That is, in 9 of the cases, the wrong critical path was reported by the estimator. A detailed analysis revealed interconnect delay to be the main source of estimator inaccuracy.

Figs. 7c and 7d shows the revised results when fanout-based delay estimation is incorporated. In this case, we use the fanouts in the CGRA device model (MRRG) as input to the estimation model. Observe that the estimation error is improved relative to the results shown above. On average, the average error is 0.73ns, 21%. After this improvement, out of the 16 benchmarks, 8 reported critical paths matched those reported by PrimeTime. Of the remaining 8 benchmarks, there are 6 cases where Tatum did report the correct paths in the respective top-four critical paths of each case.

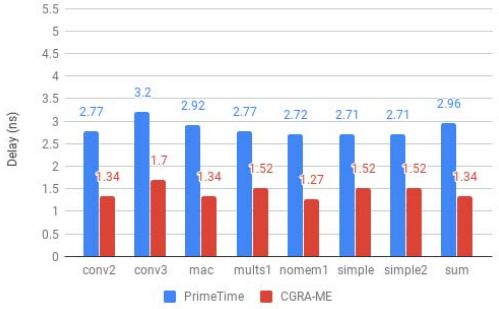
Further analysis of the results incorporating interconnect delay estimation showed that a significant source of error was for signals on the outputs of multiplexers driving the ADRES functional units. For these signals, we observed a large difference between the fanout values in the MRRG and the fanout values in the post-Synopsys netlist – i.e. after the CGRA was implemented in standard cells. As the ADRES functional units can perform a wide range of computations (add, shift, logical, multiply, etc.), the fanout of the signals in the standard-cell netlist was large – around 50-100 for each signal, versus the MRRG-based fanout of 9. In light of this fanout gap, we added the capability for a user to override the MRRG fanout, for specific CGRA instances, if desired. With the fanout of these multiplexers overriden, the results are shown in Figs. 7e and 7f. The average error reduces to 0.33ns, 9.6% vs. baseline. Moreover, after the override, 12 of the 16 benchmarks reported the correct critical paths, and 4 cases where the correct critical path is reported as one of the top-four critical paths by Tatum.

Table IV summarizes the correctness of reported critical path for all methods. $\times$'s in the table indicate that the critical path reported by PrimeTime (actual) is not reported by Tatum (estimator). $\triangle$'s indicate the PrimeTime-reported path is one of the top four paths reported by Tatum. $\bigcirc$'s represent the same path being reported in both PrimeTime and Tatum. Observe that, when fanouts are taken into account to estimate interconnect delay, there is greater alignment between the paths reported by PrimeTime and Tatum (more circles appear in the centre

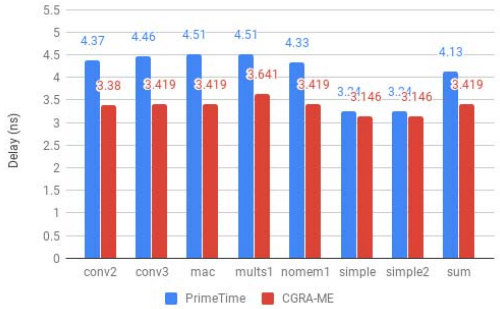
Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.



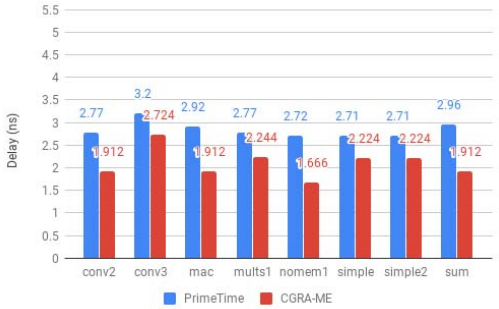
(a) Area-optimized target – without interconnect delay taken into account.



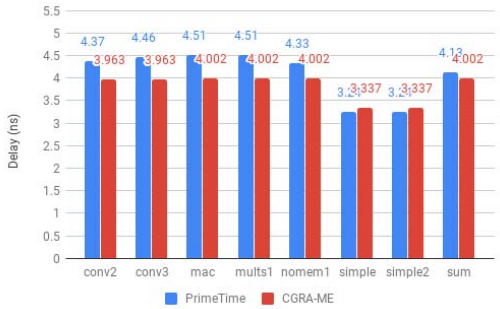
(b) Delay-optimized target – without interconnect delay taken into account.



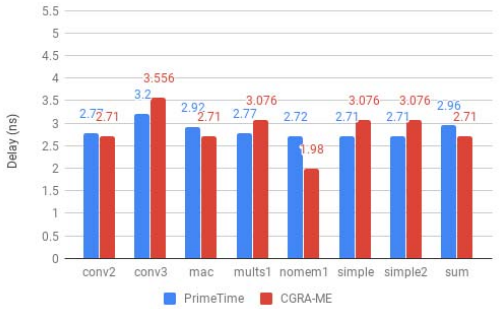
(c) Area-optimized target – with MRRG fanout-inferred interconnect delay.



(d) Delay-optimized target – with MRRG fanout-inferred interconnect delay.



(e) Area-optimized target – with selected fanout count overridden.



(f) Delay-optimized target – with selected fanout count overridden.

Fig. 7: Critical path delay comparison – CGRA-ME vs PrimeTime.

	Without Interconnect Delay		MRRG Based Fanout #		Overriden Fanout #	
Target	Area Optimized	Delay Optimized	Area Optimized	Delay Optimized	Area Optimized	Delay Optimized
conv2	✕	△	✕	○	○	○
conv3	✕	△	✕	△	○	△
mac	△	○	△	○	○	○
mults1	△	○	△	○	○	○
nomem1	○	○	○	○	○	△
simple	○	△	○	△	○	△
simple2	○	△	○	△	○	△
sum	△	○	△	○	○	○

TABLE IV: Correctness of reported critical paths: 1) ○ – Tatum reported the same path as PrimeTime, 2) ✕ – path from PrimeTime is not reported by Tatum, 3) △ – path from PrimeTime is reported as one of the top-four critical paths from Tatum.

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.

and right-most columns of the table).

In addition to the architecture presented in this study, we have applied the same methodology to another variant of the same architecture, as well as an entirely different architecture. With our estimation engine, we are able to draw relative performance conclusions among architecture variants, and even different architectures.

VI. Conclusions and Future Work

CGRA-ME is an open-source modelling and exploration framework for CGRA architecture and CAD, currently under development at the University of Toronto. In this work, we augmented CGRA-ME with capabilities for rapid area and performance estimation, without requiring time consuming full standard-cell mapping, placement, routing, extraction and timing analysis. Our area estimation is based on a characterization of primitive CGRA modules (done once offline), and storing the area figures in a library. Area estimation for a modelled CGRA is done by library lookup and aggregating the areas of used primitives. Performance estimation is more involved, requiring both primitives characterization, and the incorporation of a complete static-timing analysis (STA) engine into CGRA-ME. On average, for an ADRES-like CGRA architecture, area and performance estimates were within 7% and 10% of actual post-layout data, respectively. Moreover, the critical paths reported by the estimator are closely aligned with the actual post-layout critical paths.

Future work includes assessing the estimation accuracy on other CGRA architectures (apart from ADRES), larger grids, as well as extending the estimator to include power/energy consumption. In addition, the model can be extended to include memory interface area/performance estimation, and to handle multi-context CGRAs. We are also interested in implementing CGRAs as *overlays* on FPGAs, which, at a minimum would require repeating the primitive module characterization steps for an FPGA target, and may possibly require devising entirely new estimation approaches.

Tthe work described is released open-source as part of the CGRA-ME framework.

VII. Acknowledgements

The authors gratefully acknowledge funding from Huawei for a portion of this research.

References

[1] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *IEEE ASAP*, 2017, pp. 184–189.

[2] V. Tehre and R. Kshirsagar, "Survey on coarse grained reconfigurable architectures," *International Journal of Computer Applications*, vol. 48, no. 16, pp. 1–7, 2012.

[3] A. Chattopadhyay, "Ingredients of adaptability: a survey of reconfigurable processors," *VLSI Design*, vol. 2013, p. 10, 2013.

[4] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: ultra low-power samsung reconfigurable processor for biomedical applications," *ACM TRETS*, vol. 7, no. 3, pp. 22:1–22:15.

[5] T. Toi, N. Nakamura, T. Fujii, T. Kitaoka, K. Togawa, K. Furuta, and T. Awashima, "Optimizing time and space multiplexed computation in a dynamically reconfigurable processor," in *IEEE FPT*, 2013, pp. 106–111.

[6] S. A. Chin, K. P. Niu, M. Walker, S. Yin, A. Mertens, J. Lee, and J. H. Anderson, "Architecture exploration of standard-cell and FPGA-overlay CGRAs using the open-source CGRA-ME framework," in *ACM ISPD*, 2018, pp. 48–55.

[7] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next generation architecture and cad system for FPGAs," *ACM TRETS*, vol. 7, no. 2, pp. 6:1–6:30, June 2014.

[8] A. Chattopadhyay, X. Chen, H. Ishebabi, R. Leupers, G. Ascheid, and H. Meyr, "High-level modelling and exploration of coarse-grained reconfigurable architectures," in *IEEE/ACM DATE*, 2008, pp. 1334–1339.

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

[10] D. Suh, K. Kwon, S. Kim, S. Ryu, and J. Kim, "Design space exploration and implementation of a high performance and low area coarse grained reconfigurable processor," in *IEEE FPT*, 2012, pp. 67–70.

[11] Y. Kim, R. N. Mahapatra, and K. Choi, "Design space exploration for efficient resource utilization in coarse-grained reconfigurable architecture," *IEEE TVLSI*, vol. 18, no. 10, pp. 1471–1482, 2010.

[12] L. Yan, T. Srikanthan, and N. Gang, "Area and delay estimation for fpga implementation of coarse-grained reconfigurable architectures," *SIGPLAN Not.*, vol. 41, no. 7, pp. 182–188, Jun. 2006.

[13] 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.

[14] S. Y. Ohm, F. J. Kurdahi, N. Dutt, and M. Xu, "A comprehensive estimation technique for high-level synthesis," in *IEEE ISSS*, 1995, pp. 122–127.

[15] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 5, pp. 255–261, 2003.

[16] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to CGRA mapping," in *ACM/IEEE DAC*, 2018.

[17] "NanGate FreePDK45 generic open cell library," http://projects.si2.org/openeda.si2.org/projects/nangatelib, accessed: 2018-06-30.

[18] K. Murray and V. Betz, "Tatum: Parallel Timing Analysis for Faster Design Cycles and Improved Optimization," in *IEEE FPT*, 2018.

[19] R. B. Hitchcock, G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM Journal of Research and Development*, vol. 26, no. 1, pp. 100–105, 1982.

[20] S. Malik and H. Jyu, "Prediction of interconnect delay in logic synthesis," in *IEEE European Design and Test Conference (EDTC)*, 1995, p. 411.

[21] 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 *FPL*, 2003, pp. 61–70.

Authorized licensed use limited to: Harbin Institute of Technology. Downloaded on July 04,2023 at 22:40:51 UTC from IEEE Xplore. Restrictions apply.