2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)

Generic Connectivity-Based CGRA Mapping via Integer Linear Programming

Matthew J. P. Walker
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto
Toronto, Ontario, Canada
Email: matthewjp.walker@mail.utoronto.ca

Jason H. Anderson
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto
Toronto, Ontario, Canada
Email: janders@ece.utoronto.ca

***Abstract*—Coarse-grained reconfigurable architectures (CGRAs) are programmable logic devices with large coarse-grained ALU-like logic blocks, and multi-bit datapath-style routing. CGRAs often have relatively restricted data routing networks, so they attract CAD mapping tools that use exact methods, such as Integer Linear Programming (ILP). However, tools that target general architectures must use large constraint systems to fully describe an architecture's flexibility, resulting in lengthy run-times. In this paper, we propose to derive connectivity information from an otherwise generic device model, and use this to create simpler ILPs, which we combine in an iterative schedule and retain most of the exactness of a fully-generic ILP approach. This new approach has a speed-up geometric mean of 5.88× when considering benchmarks that do not hit a time-limit of 7.5 hours on the fully-generic ILP, and 37.6× otherwise. This was measured using the set of benchmarks used to originally evaluate the fully-generic approach and several more benchmarks representing computation tasks, over three different CGRA architectures. All run-times of the new approach are less than 20 minutes, with 90th percentile time of 410 seconds. The proposed mapping techniques are integrated into, and evaluated using the open-source CGRA-ME architecture modelling and exploration framework [1].**

I. Introduction

Coarse-grained reconfigurable architectures (CGRAs) are a class of programmable logic device where the processing elements (PEs) are large ALU-like logic blocks, and the interconnect fabric is bus-based. This stands in contrast to field-programmable gate arrays (FPGAs), which are configurable at the individual logic-signal level. CGRAs dedicate less area to flexibility/programmability, and require far fewer configuration bits than FPGAs, thereby easing CAD complexity by reducing the number of decisions tools need to make. Despite their reduced flexibility, CGRAs are an ideal media for applications where: 1) *some* flexibility is required, 2) software programmability is desired, and 3) compute/communication needs closely match with the CGRA capabilities. CGRAs can be realized as custom ASICs, or alternatively, implemented on FPGAs as overlays, and a number of commercial and academic architectures have been proposed, stretching back to the 1990s [2], [3]. With the coming end to Moore's Law, CGRAs are receiving renewed interest as platforms for domain-specific compute acceleration. As such, it is desirable to develop methodologies for the modelling and evaluation of hypothetical CGRAs. The open-source CGRA-ME (CGRA Modelling and Exploration) framework from the University of Toronto [1] aims to provide this capability.

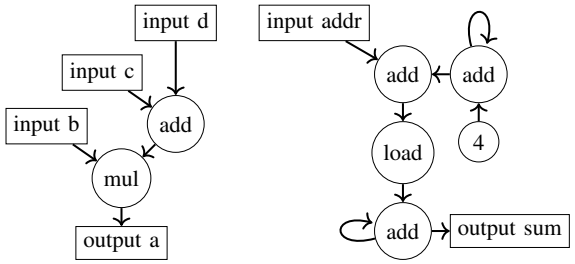
The CGRA-ME framework allows a human architect to describe a hypothetical CGRA using an expressive graph-based device model, and provides generic mapping approaches that allow an application benchmark to be mapped into the described CGRA. Of particular interest for architecture exploration is the integer linear programming-based (ILP) mapping approach, as it provides certainty regarding the mappability of an application benchmark into an architecture [4]. With such an exact mapper, an architect can be confident whether an architecture is viable for a set of applications. This approach performs well for very small architectures and application benchmarks, but does not scale very well – the runtimes of larger benchmarks and architectures can extend into the day range on a typical workstation. To truly enable architecture exploration, mapping times should be significantly, and consistently, less. It is precisely this challenge we address in this paper, namely, that of providing scalable mapping algorithms for CGRAs, while retaining the exactness property and genericity.

Through use of CGRA-ME, we have observed that mapping times are seemingly random – mapping slightly perturbed benchmarks to the same architecture may take seconds, minutes or hours. This effect becomes more pronounced with larger ILP models, the size of which is determined by both benchmark and device model size. Application benchmarks are generally quite small, and cannot be simplified. Conversely, the device model generally has thousands of vertices, leading to a large ILP problem.

The existing technique [4] retains all the flexibility of the device model, however, some of this flexibility is effectively useless. For example, many CGRAs are grid-based, and we observe that is is uncommon for the output of one PE to have a destination PE that is more than two "hops" away. Also, many CGRAs have extremely restricted routing networks, where, for example, only nearest-neighbor connectivity is present between PEs.

Given limited practical need for long routing paths, and generally non-contested connections, we present a simpler, smaller, ILP that captures most of the flexibility (Section IV). Further, PEs that are "close together", and the connections

2576-2621/19/$31.00 ©2019 IEEE
DOI 10.1109/FCCM.2019.00019



(a) `a = b * (c + d)` (b) A sum over an array of 4-byte numbers

Fig. 1. Example DFGs.

between them, can be derived from CGRA-ME's device model (Sections III-A and III-B) and the ILP formulation can be restricted to consider such information. Finally, Section III-C presents an algorithm to efficiently map benchmarks to CGRAs by using variants of the proposed ILP formulation, where we iteratively generate ILP mapping formulations that consider successively larger portions of the solution space.

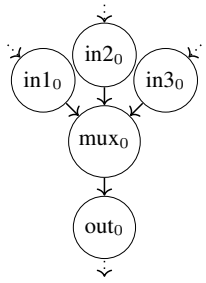
II. Background

A. Data-Flow Graphs

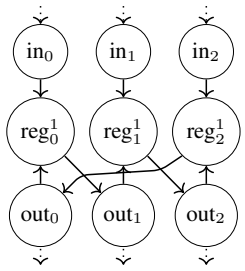
A benchmark or application kernel's essential structure can be represented as a directed hypergraph, called a *data-flow graph* (DFG), such as those in Fig. 1. In simple cases (e.g. Fig. 1a) they may be thought of as similar to abstract syntax trees, but only include values (corresponding to edges) and operations (corresponding to vertices). In more more complex cases they may have loops (e.g. Fig. 1b) and re-convergent paths (e.g. $(a + b) * (a + c)$). For the purposes of CGRA mapping, the edges are interpreted as dependency relations between operations, capturing the set of operations that must be performed before a given operation can proceed, and where data must be routed. Loads, stores, inputs, outputs and constants are also modelled as vertices, and loop-carried dependencies correspond to back-edges/loops. The input to mapping CGRA-ME is: 1) an application DFG, and 2) a device model for the targeted CGRA, called a Modulo Routing Resource Graph (MRRG), described in Section II-C.

B. Multi-Context CGRAs

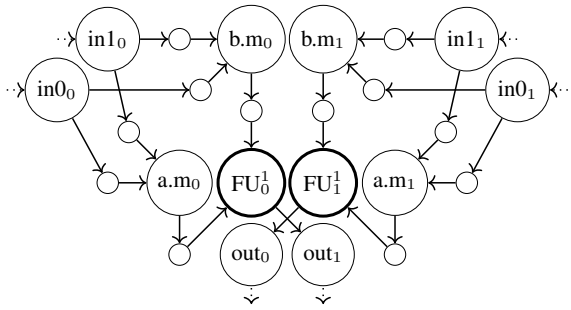
An important property of previously proposed CGRAs (e.g. [5]–[8]) is the notion of *multiple contexts*. A context is a single configuration of the CGRA's logic functionality and routing connectivity. A two-context CGRA would contain *two* copies of its configuration cells: configuration 0 and 1. The typical behavior of such a CGRA is to cycle between the two contexts on a cycle-by-cycle basis. This implies that the hardware functionality and routing can change each cycle. The hardware is thus "time multiplexed", where PEs and routing can be used for different purposes in each context. A PE can, for example, perform an addition in context 0, store the result at the clock edge, and then perform a multiply in context 1. This is as opposed to today's commercial FPGAs, which are



(a) A 3-to-1 mux for II = 1. The structure would be duplicated for higher II, with appropriate changes to the t values.



(b) A register for II = 3. Note the edges that connect from context 0 to context 1, 1 to 2 and 2 to 0.



(c) A simple processing element for II = 2, consisting of a two-input latency-one ALU with one two-input mux on each input (".m" nodes), forming a crossbar from the input nodes to the FU nodes. The attachment points at the "out" nodes would typically connect to a register like Fig. 2b.

Fig. 2. Example MRRG fragments. Subscripts indicate the value for *t*, and superscripts indicate the latency, if any. Dotted arrows indicate points where these fragments would get attached to the rest of the MRRG. Nodes that may have an operation mapped to them have thick outlines.

single context. A typical CGRA has a range of configuration context counts that it can physically realize, and a mapping tool will typically try to minimize this, as it is equal to the initiation interval (II) – the rate at which new inputs are consumed by the CGRA.

C. Modulo Routing Resource Graphs (MRRGs)

An MRRG [5] is a graph data structure that is commonly used to model the hardware connectivity and capability of CGRAs [9], [10]. It is used as the CGRA device model within the CGRA-ME framework [4]. An MRRG is a directed graph where a vertex is a hardware element in time and space, and an edge represents a possible fanout. In the variant used by CGRA-ME, a vertex is a 2-tuple (s, t) with physical node id s, and context number t (**t**ime). Edges represent connectivity across time and space, and include connections that "wrap around" from t to $t' \leq t$ (eg. Fig. 2b). This is sufficient to describe the structure, but to capture the entire behavior some extra data is tagged on each vertex, such as latency and its supported computation operations, if any. We will refer to nodes that support computation operations as functional unit (FU) nodes.

In Fig. 2a we have a single context MRRG fragment that is used to represent a multiplexer hardware element. Because there is no latency associated with it, all connections between vertices are within the same context, 0. In contrast, Fig. 2b is used to express a register of configurable latency in a multi-context CGRA and contains connections between contexts.

A typical implementation of a PE will contain one FU node connected to registers and input crossbars, with the whole structure duplicated for each context, such as Fig. 2c. PEs will also typically have another FU node that provides the "constant" operation, corresponding to constants in the DFG. The inputs and output nodes of PEs are then connected together according to the connectivity of the architecture. To provide support for DFG inputs and outputs, typical CGRA models will also have several FU nodes that implement only these IO operations.

To keep mapping generic, even though the MRRG is formed from many stitched together graphs like those in Fig. 2, CGRA-ME ignores this and only uses the connectivity of the flattened graph [4].

D. Integer Linear Programming

Integer linear programming (ILP) is a powerful and generic tool for specifying and solving combinatorial optimization problems. An ILP consists of three parts: a set of integer variables, a set of inequality constraints on weighted sums of the variables, and optionally a cost function (another weighted sum) to choose the best solution. Once these are specified, one of many free or commercial solvers can be used to find a solution. A solver will always find the optimal solution given enough time, but in general ILP is NP-complete, and solve times may be lengthy.

E. CGRA-ME's Existing Approach

The entire problem of mapping to a CGRA can be described as taking a DFG that describes the computation and "finding it" in the MRRG that describes the hardware. This amounts to matching vertices in the DFG with MRRG vertices that support the operation, and edges in the DFG with paths in the MRRG. Specifically, this is very similar to the directed sub-graph homeomorphism problem [11], except that paths starting at the same MRRG vertex may initially overlap, as the DFG is a hypergraph. The existing approach, due to S. A. Chin [4], directly encodes this problem in an ILP, with the general approach being: *if an MRRG node is used by a particular DFG edge, then at least one of its fanout must be too*. This maxim is applied to every node, for every DFG edge, resulting in $\mathcal{O}(|E(\text{DFG})| \cdot |V(\text{MRRG})|)$ constraints. Additionally, it requires that if two MRRG nodes are connected by an edge, at most one of them may have multiple fanins. This is to implement a constraint to prevent self reinforcing loops in the mapping: a cycle of only routing nodes satisfies the maxim above. While this approach generally results in a very large ILP, it is guaranteed to capture 100% of the flexibility of the architecture.

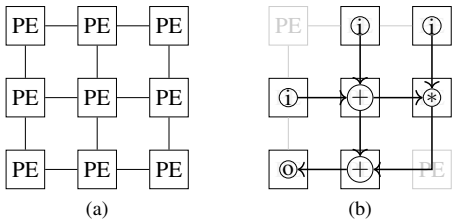


Fig. 3. (a): A simple, orthogonally connected, CGRA. Any two adjacent PEs may be selected for ALU inputs, and each PE may simply route-through one input instead of using its ALU. And (b): an example mapping of DFG that requires use of a route-through.

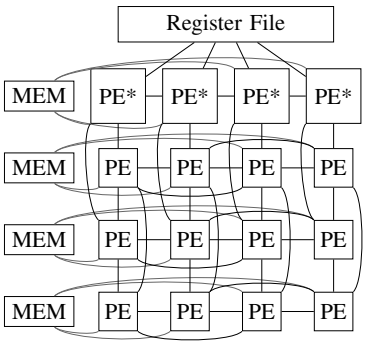


Fig. 4. ADRES [6] architecture. A heterogeneous array of a fully-featured top row of processing elements, with supporting PEs underneath that can only perform addition and subtraction. PEs are connected to vertically and horizontally adjacent neighbor, and distance-two neighbor. IO is done through the register file, and route-throughs are supported.

III. Connectivity-Based CGRA Mapping

A. Connectivity

In CGRAs, the small number of input ports on a ALU (usually 2) means that every processing element can have a fully-connected crossbar – even in CGRAs with up to 8 PE inputs [6] (ADRES) or very flexible routing [8] (HyCUBE). As a prototypical example, consider the orthogonally-connected CGRA in Fig. 3a. A connection between adjacent processing elements is guaranteed. If there is minimal need to negotiate routing, there should be a corresponding minimal need to explicitly model all the individual multiplexers and connections. In this work we find this to be the case, even for ADRES or HyCUBE, though less so for our "Clustered" architecture (Figs. 4 to 6). These architectures have similar computation capability, but differ primarily in how the PEs are connected to each other; HyCUBE is the most flexible, and Clustered and ADRES are less so. Clustered has small links between islands of fully-connected PEs, while ADRES has folded-torus connectivity.

Further, given that we observe it is uncommon for the output of a PE to drive another PE that is far away, we present an ILP based around a more abstract principle: *if an operation is mapped to a PE, then the fanin of the operation must be mapped to neighboring PEs* (constraints 1 to 4 below). Any choice of definition for a PE's neighbor

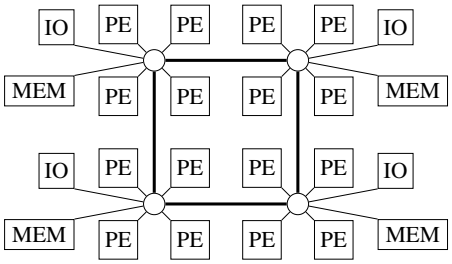


Fig. 5. A clustered architecture. Groups of 4 PEs connected to crossbars connected in a grid. PEs within a cluster are fully-connected, but the number of connections between clusters is limited to 2. Each cluster also has one memory port and one IO port.

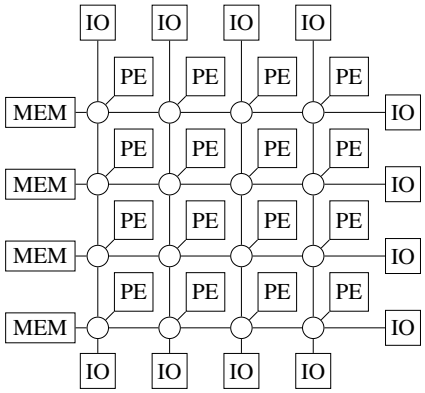


Fig. 6. HyCUBE [8] Arch. A homogeneous array of PEs, each associated with a fully-connected crossbar which are connected in a grid.

will work, but we opted for a heuristic method in this work: a symmetric breadth-first search in the MRRG. To find the neighbors of one FU node, a breadth-first search is started at that node, and after every expansion wave, the number of FUs discovered is compared to the target number-of-neighbor. If at least the target number-of-neighbor is found, then the search terminates and neighbor are recorded. This search is repeated for each FU node in the graph using the same target number-of-neighbor, depending on the stage of the algorithm (details in Section III-C). This is can be thought of as using a "contracted" MRRG: routing between "neighboring" FU nodes is eliminated, leaving only FUs connected to other FUs.

The target number-of-neighbor to search for before constructing the ILP must be carefully chosen. Consider Fig. 3b: there is no way to map the DFG to this CGRA without using another common feature of CGRAs: a PE route-through (see the lower-right PE in the figure). Many CGRAs can convert a PE into a route-through, though others may provide features such as diagonal, torus and/or skip connections. The number-of-neighbor that should be found in order to allow a DFG to be mapped is therefore architecture- *and* DFG-dependent. Effects of choosing particular number-of-neighbor are discussed in Section V-A.

B. Paths

For simple architectures like Fig. 3a, it may be pointless to model routing congestion, but for more complicated architectures such as Figs. 5 and 6 we find it beneficial to model routing congestion.

From an intuitive view of the clustered architecture in Fig. 5, it is obvious that in this CGRA there are limited connections between clusters. However, deriving neighbors from a breadth-first search is oblivious to this. In fact, if we do not model routing, we find that for architectures like this, we must iterate through many placements before finding one that can route. To include routing in the ILP, we follow two principles: 1) *if an FU drives another FU, then a path through the MRRG from the driver to the driven must be chosen*; and 2) *two paths that are driven by different FUs cannot share a vertex* (constraints 5 and 6 below, respectively).

The potential downside of choosing paths from a finite set is that some of the combinatorial expressiveness of a graph-based device model can be lost. Consider that between two FUs there may be a number of crossbars. The number of paths between these two FUs is at least equal to the product of the widths of the crossbars. The method used in this work is to find the 20 cycle-less shortest paths between each pair of neighboring FU nodes, as this works for the 3 architectures under study.

C. Composition

As a first pass, a reasonable approach is to choose a sufficiently high number-of-neighbor to search for, identify a selected number of paths between each FU, and then try to solve a combined placement and routing ILP. We found this to work well for trivially small architectures, but the large number of variables required for choosing paths results in an ILP with solve times greater than 20 minutes for most benchmarks on all 3 architectures in this work.

Alternatively, placement and routing can be completely split up by not modelling paths at all when finding a placement ("placement-only"), and then testing if a placement can be routed by a "routing-only" ILP that chooses paths for the connections required by the placement. This approach has small ILPs for each stage, but the the placement-only ILP has no guidance as to routability.

To address this, one solution is to choose an ILP cost function for the placement that reflects how reliably a route can be found between two processing elements. Another solution is to add a form of congestion modelling by adding the constraints for choosing paths, but relaxing the number of paths that may use the same vertex (i.e. allowing shorts). Implementing either of these approaches requires adding variables that model whether a pair of FUs are used (variables $\mathbf{e}$ below), and result in similarly sized ILPs. In this work, we use the "relaxed-constraint-placement" because it overall takes less time to discover a placement that will route – even when compared to the cost-based approach with a cost function that heavily favours edges between FUs that are close together in the MRRG. The quality of placements in either case is similar, but the addition of a cost function slows down the

rate that solutions are discovered significantly. We find that relaxing the vertex usage to at least 2 is effective, and that with this relaxation, we can reduce the number of paths for each connection to at least 3.

This relaxed-constraint-placement produces an ILP big enough that the solver cannot consistently determine feasibility in less than $200\,\mathrm{ms}$. For example, the number-of-neighbor parameter may be too low to map the DFG, and the solver may take several tens of seconds to prove infeasibility. However, we also aim to keep the number-of-neighbor as low as possible: a higher number results in an ILP with larger constraints and more solutions – many of which will be unroutable, e.g. requiring conflicting route-through usage. Our strategy is to characterize an architecture to determine a schedule of number-of-neighbor to try, and use the the placement-only ILP as a test for a given number-of-neighbor. Its small model size makes it an low-runtime solution, and we find that even though this placement-only ILP does not model congestion, it is nearly 100% predictive of whether a solution exists to the larger relaxed-constraint-placement ILP. Still, there are occasionally many solutions, with none routable, so we limit the number placements to 100 before moving on to the next number-of-neighbor. The proposed algorithm is the following:

```
for nn in CGRA.neighborCountSchedule():
  if placementOnly(nn).failure():
    continue;
  for pment in relaxedConstrPment(nn)[0:100]:
    routing = routingOnly(pment);
    if routing.success():
      return pment + routing;
return not_mappable;
```

IV. ILP Formulation

The formulations discussed so far can be expressed as one ILP, with certain sets of constraints removed or relaxed to create the specific formulations. The ILPs use the following definitions:

- FU $\subseteq V(\text{MRRG})$: all MRRG nodes that can perform computation.
- $\text{comp}(o \in \text{FU})$: the set of MRRG FU nodes that o can be mapped to.
- $\text{neigh}(u \in FU)$: a set of MRRG FU nodes that are reachable from u; Section III-A's graph search results.
- $\text{paths}(u \in \text{FU}, v \in \text{FU})$: a set of paths through the MRRG from u to v; Section III-B's results.

The following variables describe the mapping:

- $\mathbf{f}_{ou}$: is FU u used By DFG operation o?
- $\mathbf{p}_{uvq}$: is path number q used from FU u to v?

Also, a necessary intermediate variable class is used:

- $\mathbf{e}_{oupv}$: is the DFG edge (o,p) mapped to FUs u to v?

Fig. 7 visualizes the relationship between the FU-is-used variables ($\mathbf{f}$) and edge-is-used variables ($\mathbf{e}$). The placement-only ILP uses all constraints described below, except 5 and 6. The routing-only ILP only uses constraints 5 and 6, with

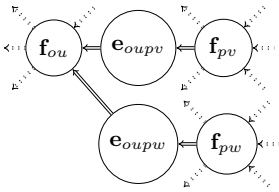


Fig. 7. Visualisation of the relation between edge and FU variables. Arrows indicate implication.

$\mathbf{e}$ variables set to fixed values. Finally, the relaxed-constraint-placement ILP uses all constraints as stated, except 6, which is relaxed as described below.

1) Functional Unit Exclusivity This constraint ensures that each physical functional unit is not occupied by multiple DFG vertices.

$$\forall u \in \text{FU} \quad \sum_{o \in \text{DFG}} \mathbf{f}_{ou} \leq 1$$

2) Must Map Ops Requires that a mapping be found. Due to later constraints, it is sufficient to only apply this constraint for output nodes, specifically, a set DFG vertices whose combined fanin cones cover the entire DFG.

Also, if this constraint is relaxed to be greater-than-or-equal-to 1 (instead of exactly equal to 1), then the ILP supports duplication of operations, i.e. re-computation. The effects of this are not explored in this work. We suspect it may be useful for certain DFGs and architectures, but that it also may hinder finding useful mappings by increasing the number of unwanted mappings (such as filling the architecture with redundant computation).

$$\forall o \in \text{outputs(DFG)} \quad \sum_{u \in \text{comp}(o)} \mathbf{f}_{ou} = 1 \quad (\text{or} \geq 1)$$

3) Fanin Required This constraint encodes the notion that if an operation is mapped to an FU, then each DFG fanin of the operation must be mapped to a neighboring FU, i.e. $\mathbf{f}_{pv} \Rightarrow \exists u : \mathbf{e}_{oupv}$ for each fanin o. This constraint ensures that all data that are needed by FU v will arrive.

$$\forall (o,p) \in \text{DFG}\ \forall v \in \text{comp}(p)$$
$$\mathbf{f}_{pv} \leq \sum_{u \in \text{comp}(o) \text{ if } v \in \text{neigh}(u)} \mathbf{e}_{oupv}$$

4) Fanout Implies Usage If an edge variable originates at a given FU, then that FU must be in use by the fanin operation, i.e. $\mathbf{e}_{oupv} \Rightarrow \mathbf{f}_{ou}$.

$$\forall (o,p) \in \text{DFG}\ \forall u \in \text{comp}(o)\ \forall v \in \text{comp}(p) \cap \text{neigh}(u)$$
$$\mathbf{e}_{oupv} \leq \mathbf{f}_{ou}$$

5) Path Required for an Edge Simply, if an edge is in use, then at least one path corresponding to it must be in use, i.e. $\mathbf{e}_{oupv} \Rightarrow \exists q : \mathbf{p}_{uvq}$.

$$\forall (o,p) \in \text{DFG}\ \forall u \in \text{comp}(o)\ \forall v \in \text{comp}(p) \cap \text{neigh}(u)$$
$$\mathbf{e}_{oupv} \leq \sum_{q \in \text{paths}(u,v)} \mathbf{p}_{uvq}$$

6) Paths are Mutually Exclusive if Driven by Different FUs If a path through the MRRG is mapped to a DFG edge, then it electrically cannot overlap with another path, unless both paths are driven by the same physical FU.

$$\forall u, v, w, x \in \text{FU}\ \forall q \in \text{paths}(u, v)$$
$$\forall z \in \{\text{paths}(w, x) : \neg\, \text{pcompat}(u, q, w, z)\}$$
$$\mathbf{p}_{uvq} + \mathbf{p}_{wxz} \leq 1$$

Where $\text{pcompat}(u, q, w, z) = (q \cap z \stackrel{?}{=} \varnothing) \vee (u \stackrel{?}{=} w)$, i.e. do the paths not overlap, or are they driven by the same FU node. By relaxing this constraint to be less than or equal to some integer greater than 1, an ILP that allows routing overuse is created. Also, the constraint as-specified will produce duplicate constraints, which are detected and not added to the ILP.

For ILPs that model congestion or routing, the variables that are required by this constraint category will easily dominate the number of variables attributed to other categories. More precisely, the number of $\mathbf{p}$ variables is linear with respect to the CGRA size, number-of-neighbor used, and number of paths used – $\mathcal{O}(|\text{FU}| \cdot N \cdot P)$, so the number possible conflicts is in $\mathcal{O}(|\text{FU}|^2 \cdot N^2 \cdot P^2)$.

7) With a Generic Cost Function With variables representing each aspect of a mapping, various cost functions can be specified. Using the first summation below, the coefficients k_{ou} can specify that a certain FU node u for DFG vertex o is preferred over other FU nodes, such as to encourage using a certain portion of the CGRA. The center summation can be used to select certain FU-FU connections to be preferentially chosen for particular DFG edges, or, that using certain pairs of FU nodes is preferred over others. For example, if coefficients l_{oupv} are set to the taxicab distance between the FUs u and v, then the half-perimeter bounding box is minimized. The bottom summation can be used to encourage the choice of particular paths, such as by setting m_{uvg} to the estimated power consumption of using that path.

$$\mathcal{L} = \sum_{o \in \text{DFG}} \sum_{u \in \text{comp}(o)} k_{ou} \mathbf{f}_{ou}$$
$$+ \sum_{(o,p) \in \text{DFG}} \sum_{u \in \text{comp}(o)} \sum_{v \in \text{comp}(p) \cap \text{neigh}(u)} l_{oupv} \mathbf{e}_{oupv}$$
$$+ \sum_{u,v \in \text{FU}} \sum_{q \in \text{paths}(u,v)} m_{uvq} \mathbf{p}_{uvq}$$

V. Experimental Results

A. Setup & Overview

To simulate various sizes of architecture, and to demonstrate variability in runtime, we test each architecture with 1, 2 and 3 CGRA contexts. We use the set of benchmarks [4] used to characterize the existing CGRA-ME approach, plus some additional computation applications, and a fast-Fourier-transform benchmark from MiBench [12]. We are limited by our range of benchmarks because our DFG generator does not support conditional statements [1]. Each benchmark-architecture-II combination is run 6 times using different ILP solver seeds for robust runtime measurements, and with a limit of 7.5 hours – near the length of a typical workday.

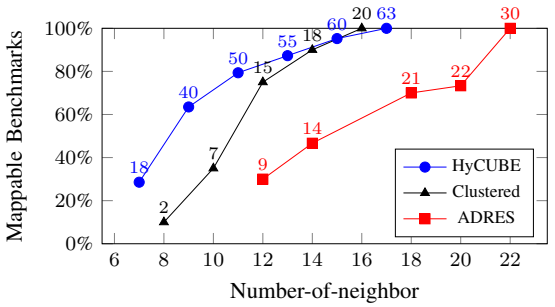


Fig. 8. Number of benchmarks that will map for a given number-of-neighbor, normalized to the number of benchmarks that will map to that architecture. Different values of II are presented as the same architecture.

The algorithm from Section III-C is used, with the schedules for number-of-neighbor determined empirically for each architecture, and presented as the horizontal values of the curves in Fig. 8. We start with the minimum value, and increment by 2 until the maximum. With these schedules, this approach provides the same feasibility results as the existing approach, whenever the existing approach can provide a decision within the time limit. Fig. 8 also shows the utility of starting at a small number-of-neighbor and increasing slowly: many benchmarks can be mapped without reaching the maximum value. The differences between architectures is also notable. HyCUBE will map many benchmarks at a low number-of-neighbor due to it flexible routing, however some benchmarks require long-distance connections. Clustered can map suddenly many more benchmarks once inter-cluster FUs are consider neighbors, but soon runs into the limits of the architecture. ADRES is much more homogeneous than the other two architectures, resulting in a more linear trend. We also suggest a generic schedule for new architectures: start at 4 with step 2, ending at 24.

The time taken to run the n-shortest-paths algorithm to find paths has been removed from the runtime of our approach by caching all data in memory before commencing the timer. Filling this cache with 20 paths between every pair of FU nodes (excessive, but guaranteed to cover all possibilities) using a single CPU thread takes at most 5.4 seconds for the largest architecture tested, HyCUBE with II = 3. Each run of the n-shortest-paths algorithm takes very little time and is completely independent, so parallelism could trivially be used to reduce this time. And, the paths discovered do not change for a given architecture and II, so they could be cached to disk.

The ILP solver used for both is the one provided by Gurobi Software [13], and experiments were performed using up to 4 threads on Intel® Xeon® Gold 6148 Processors.

B. Comparison & Discussion

Twenty-eight benchmarks over 3 architectures times 3 II values is too many data-points to present directly, so summaries of runtime data are presented instead, in Tables I and II. We

take geometric means across architecture-II or benchmark axes and compute the relative speedup. Maximum runtimes of this work are also presented. At the left of Table I, there is the benchmark that is selected for each row, presented along with the number of vertices in the benchmark DFG. To compute column 3, the average runtime over the seeds is determined for each architecture and II, and the geometric mean of the averages is taken. The computation for column 4 is exactly the same, except runtimes for this work are used. The speedup column is simply the value of column 3 divided by column 4. Column 6 is similar to column 4, except the maximum of the runtime averages is taken, instead of geometric mean. At the far right is the median number-of-neighbor required to map this benchmark on the architecture and II variants. Table II is similar, except the statistics are taken across all benchmarks while fixing architecture and II.

Looking at Table I, it is clear that certain benchmarks tend to produce long runtimes in the existing approach. However, there is no clear correlation between DFG size or complexity, with one of the largest and most complicated benchmarks (FFT) having a short average runtime compared to some smaller, simpler DFGs (eg. conv2, accumulate). This could be attributed to the solver quickly determining that there are not enough FUs to map FFT on II = 1 variants, which is the case, however conv2 and accumulate, which are able to map with II = 1, cause timeouts on II > 1 architectures.

Looking at the average runtime for this work in Table I, we see that there is a again not much of a correlation between with DFG size. However, looking at the median number-of-neighbor required to map the benchmarks there is a correlation: as the number of path exclusivity constraints increases with the square of the number-of-neighbor, a benchmark that requires a high number of fanout (exponential-*,cosh-4,cosh-4) or connections to distant FU nodes (FFT, long-chain) will take more time to map.

Here, we observe a more straightforward pattern in the existing approach: an increasing II (which essentially acts as a multiplier on MRRG size) results in increasing runtime. This is also true for this work, as a higher II implies more FU nodes which implies again that more path variables must exist.

When considering each benchmark-architecture-II experiment individually, the speedup in geometric mean of all datapoints is 37.6×. The arithmetic mean of speedups is 861×, and the median speedup is 41.7×. The 90th percentile average runtime for the new approach is 410 seconds.

For problems that the existing approach can provide an mapping (or infeasible result) within the time limit, this approach matches for all all but 17 (6.7%). All non-matching results are cases where this approach is not able to produce a mapping and the existing approach can. Fifteen of the non-matching results are for II = 1, suggesting that this approach has some trouble with highly constrained problems. Also, for ADRES and HyCUBE, the relaxed-constraint-placement ILP produces the same feasible/infeasible result as the existing approach. For architecture exploration, this may be sufficient and if used, would have a 244× speedup over the existing approach. We believe that the reason that the relaxed-constraint-placement ILP finds a solution on the Clustered architecture is that allowing shorts does not model the limited connectivity between clusters.

When there is a solution to the relaxed-constraint-placement ILP, but it is not possible to map (ie. Clustered) the runtime is entirely attributable to constructing and solving the 100 routing-only ILPs per number-of-neighbor, which may take up to 3 seconds each. This also occurs to a lesser degree on the other architectures when a benchmark is only barely not routable at some NN.

Finally, Table III presents some sample MRRGs' vertex set sizes with corresponding ILP statistics. The largest ILP used by this work is the relaxed-constraint-placement step. Numbers of constraints and variables for this ILP are presented adjacent to the same statistics for the (single) ILP in the existing approach. For the smaller, simpler architectures (ADRES & Clustered), the ILP sizes are similar. We believe that the reason solving this ILP is faster than the existing approach is the inherent flexibility of allowing shorts: an initial guess for any variable is more likely to result in a solution. We also note that this work scales better: while the constraint numbers for the existing approach follow the trend of the MRRG size, this approach remains more constant, as its more dependent on number of FUs. Models for mapping to HyCUBE are significantly smaller than ADRES and Clustered, due to it requiring a lower number-of-neighbor, which results in many fewer path variables and associated constraints. Statistics for the placement-only and routing-only ILP of this work are also included underneath, and are one to two orders-of-magnitude less than the relaxed-constraint-placement ILP.

VI. Related Work

There have been a few proposed integer linear programs for CGRA mapping [4], [14], [15], and a SAT-based one [16]. CGRA-ME's existing approach [4] as well as [15] are the most general, deferring all scheduling and mapping decisions to the ILP/SAT solver, and modelling routing explicitly. All these approaches attempt to completely retain the flexibility of the architectures that they support, but also suffer from long runtimes. In the case of [16], it was necessary to break up the SAT problem in time, so that the runtime remained bounded, via an interesting sliding-window approach. Our approach builds upon some ideas from these works, applying the new idea of a systematically simplified device model.

While the ILP and SAT methods mentioned above try to solve placement and routing at the same time, this work splits them up somewhat, like [5], [14], [17], [18] or typical FPGA mapping approaches. The aim is to break-up one intractable problem into smaller tractable problems, but CGRAs have resisted this by having inflexible routing, as evidenced by the long runtimes of [5], [17]. The approach presented here is more of a hybrid, explicitly modelling hard routing constraints during placement, and then checking feasibility in a detailed routing step.

TABLE I
GEOMETRIC MEANS OF SIX-SEED ARCHITECTURE ARITHMETIC MEANS, AND MAXIMUM RUNTIMES, BY BENCHMARK

Benchmark Name	DFG Size	Existing [4]	This Work	Speedup	This Work Max.	Median NN
cap	24	.6	.3	2.3	.6	—
multiply-16	32	18.9	2.2	8.6	591.4	10
multiply-14	28	16.4	2.2	7.4	374.5	9
nomem1	6	117.4	2.3	51.3	27.1	8
add-16	32	19.3	2.3	8.4	583.9	10
weighted-sum	32	20.9	2.3	8.9	595.7	10
add-14	28	17.5	2.4	7.4	412.2	9
sum	7	82.0	3.8	21.6	50.0	9
add-10	20	93.9	4.1	22.9	175.7	9
multiply-10	20	89.3	4.3	20.7	230.3	9
mac	11	401.4	5.3	76.2	56.7	10
simple	12	2,674.1	7.0	383.0	55.2	10
simple2	12	2,551.5	7.1	361.8	53.5	10
matrixmultiply	17	1,371.0	10.0	136.9	179.0	10
long-chain	35	29.2	10.7	2.7	1,368.1	13
FFT	38	242.1	11.4	21.2	1,680.3	—
conv2	16	3,091.3	13.3	233.2	77.8	10
accum2	18	6,971.5	20.2	345.1	242.9	9
accumulate	18	11,046.9	20.7	534.9	235.6	9
conv3	24	5,469.7	31.6	172.9	282.8	10
exponential-4	13	509.1	35.6	14.3	266.0	13
exponential-6	26	1,270.5	39.5	32.1	903.6	—
taylor-series-4	15	1,065.3	47.4	22.5	291.3	12
mac2	24	15,372.4	52.7	291.7	461.8	14
mults2	25	19,866.9	86.2	230.5	429.2	15
cosh-4	21	2,449.8	119.6	20.5	561.7	15
cos-4	21	2,436.7	119.9	20.3	583.8	16
exponential-5	19	2,295.8	135.4	17.0	522.4	15

TABLE II
GEOMETRIC MEANS OF SIX-SEED BENCHMARK ARITHMETIC MEANS, AND MAXIMUM RUNTIMES, BY ARCHITECTURE

			This Work		
Arch. Name	II	Existing [4]	Time	Speedup	Max. Time
adres-4x4	1	4.4	1.5	3.0	461.8
hycube-4x4	1	79.6	4.3	18.6	522.4
clustered-2x2	1	7.8	5.2	1.5	583.8
adres-4x4	2	310.2	7.1	43.5	728.0
hycube-4x4	2	8,422.3	13.5	625.9	1,368.1
adres-4x4	3	1,303.7	15.6	83.5	1,680.3
hycube-4x4	3	12,788.1	23.2	551.7	1,023.9
clustered-2x2	2	871.2	24.7	35.3	723.9
clustered-2x2	3	2,902.5	72.1	40.2	876.8

TABLE III
SAMPLE MRRG SIZES & CORRESPONDING ILP SIZES AFTER PRESOLVING FOR TAYLOR-SERIES-4 BENCHMARK, WITH II = 2

		No. Constraints		No. Variables	
Arch.	MRRG Size	Existing	This Work	Existing	This Work
ADRES	2320	17567	19159	18206	27472
Placement-only			423		336
Routing-only			8525		410
Clustered	4488	22409	19247	22002	25728
Placement-only			511		384
Routing-only			6931		412
HyCUBE	5056	52995	12947	42208	17896
Placement-only			599		432
Routing-only			4421		412

This approach is influenced by [9], [19]–[22], in that it primarily operates on connectivity information, but these other mapping procedures are each tuned/designed for a specific class of architectures. The methods of [14], [18]–[20] manipulate the DFG to assist in mapping the architectures (such as node duplication, and adding explicit "routing" nodes). To remain generic, we hesitate to manipulate the DFG, instead allowing scheduling to be decided as part of the placement.

VII. CONCLUSION & FUTURE DIRECTIONS

We have presented an ILP-based method for mapping applications to a variety of CGRAs in a reasonable amount of time, with significant improvement over the state-of-the-art in generic CGRA mapping [4]. To extend this work, the derived connectivity information could instead be applied to a polynomial-time approach generalized from one of [19]–[21]. A separate direction is to not solve the entire problem at once: partition the DFG and/or CGRA, or use the "sliding window" approach of [16]. However, partitioning and incremental techniques are difficult to do generically, and must be done with care to take advantage of the capabilities of the architecture. As discussed in Section V-B, reducing the number of path variables is essential to the performance of this work, so heuristic methods for achieving this are of interest.

ACKNOWLEDGEMENT

The authors gratefully acknowledge Huawei for supporting this research. Computations were performed on the Niagara supercomputer at the SciNet HPC Consortium. [23]

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] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in *ACM/IEEE DATE*, 2001, pp. 642–649.

[3] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, pp. 332–354, 2015.

[4] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to CGRA mapping," in *ACM/IEEE DAC*, 2018, 128:1–128:6.

[5] B. Mei, S. Vernalde, D. Verkest, H. D. 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, Sep. 2003.

[6] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *FPL*, P. Y. K. Cheung and G. A. Constantinides, Eds., 2003, pp. 61–70.

[7] K. Patel, S. McGettrick, and C. J. Bleakley, "SYSCORE: A coarse grained reconfigurable array architecture for low energy biosignal processing," in *IEEE FCCM*, May 2011, pp. 109–112.

[8] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *ACM/IEEE DAC*, 2017, 45:1–45:6.

[9] 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 *ACM/IEEE PACT*, 2008, pp. 166–176.

[10] L. Chen and T. Mitra, "Graph minor approach for application mapping on CGRAs," *ACM TRETS*, vol. 7, no. 3, 21:1–21:25, 2014.

[11] S. Fortune, J. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoretical Computer Science*, vol. 10, no. 2, pp. 111–121, 1980.

[12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization*, Dec. 2001, pp. 3–14.

[13] Gurobi Optimization. (2019). Mixed integer program solver, [Online]. Available: http://www.gurobi.com/.

[14] G. Lee, K. Choi, and N. D. Dutt, "Mapping multi-domain applications onto coarse-grained reconfigurable architectures," *IEEE TCAD*, vol. 30, no. 5, pp. 637–650, 2011.

[15] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, "A general constraint-centric scheduling framework for spatial architectures," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 495–506, Jun. 2013.

[16] S. Chaudhuri and A. Hetzel, "SAT-based compilation to a non-vonNeumann processor," in *IEEE/ACM ICCAD*, 2017, pp. 675–682.

[17] 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.

[18] S. Das, T. Peyret, K. Martin, G. Corre, M. Thevenin, and P. Coussy, "A scalable design approach to efficiently map applications on CGRAs," in *IEEE Symp. on VLSI*, 2016, pp. 655–660.

[19] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *ACM/IEEE DAC*, 2012, pp. 1280–1287.

[20] ——, "REGIMap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *ACM/IEEE DAC*, 2013, 18:1–18:10.

[21] W. Kim, D. Yoo, H. Park, and M. Ahn, "SCC based modulo scheduling for coarse-grained reconfigurable processors," in *IEEE FPT*, 2012, pp. 321–328.

[22] 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 *IEEE Conference on Design and Architectures for Signal and Image Processing*, IEEE, 2010, pp. 168–175.

[23] C. Loken, D. Gruner, L. Groer, R. Peltier, N. Bunn, M. Craig, T. Henriques, J. Dempsey, C.-H. Yu, J. Chen, L. J. Dursi, J. Chong, S. Northrup, J. Pinto, N. Knecht, and R. V. Zon, "SciNet: Lessons learned from building a power-efficient top-20 system and data centre," *Journal of Physics: Conference Series*, vol. 256, 2010.