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, CAD 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 deriving certain properties from an otherwise generic device model, and using these properties to create a simpler ILP that retains most of the exactness of a fully-generic ILP. This approach has a speed-up geometric mean of 85× when considering benchmarks that do not hit a time-limit of 7.5 hours on the fully-generic ILP, and 494× 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 10 minutes, with 90th percentile time of 7.9 seconds, while being more robust to seed effects in an ILP solver, due to the smaller model size. 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 retain 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, leaving only the device model under scrutiny. And, we find that the graph-based device model, while extremely expressive, generally has thousands of vertices, directly leading to a large ILP problem.

The existing technique [4] retains all 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-neighbour 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, in Section IV. Further, PEs that are “close together”, and the connections 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 graph, 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 complex cases they may have loops (e.g. Fig. 1b) and re-convergence. 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 a mapper in CGRA-ME is an application DFG, and a device model for the targeted CGRA, called a Modulo Routing Resource Graph (MRRG), described below.

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 configuration 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 underlying 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 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 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 (time). Edges represent connectivity across time and space, and include connections that “wrap around” from t to t’ ≤ t (eg. Fig. 2b). This is sufficient to describe the structure, but to capture the entire behaviour some extra data is tagged on each vertex, such as latency and supported computation operations, if any. We will refer to nodes that support computation operations as functional unit (FU) nodes. A typical implementation of a PE will contain one FU node for computation, with a register sub-graph attached, surrounded by input and output crossbars, PEs will also typically have another FU node that only can provide the “constant” operation, corresponding to constants in the DFG. To provide support for DFG inputs and outputs, CGRA models will also have several FU nodes that implement only these IO operations.

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 variable latency in a multi-context CGRA and contains connections between contexts. Fig. 2c presents a nearly complete graph for a 2-input processing element with an input crossbar.

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 solutions. A solver will always find the optimal solution given enough time, but in general ILP is NP-complete, and solve times may be lengthy. The solver used in this work for both the existing CGRA-ME mapping approach and the new one presented is Gurobi Optimization’s solver [11].

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 subgraph homeomorphism problem [12], except that paths starting at the same MRRG vertex may initially overlap. The existing approach [4] directly encodes this problem in an ILP, with the general approach being if a MRRG node is used by a particular DFG edge, then at least one of it’s fanout must be too. This maxim is applied to every node, for every DFG edge, resulting in $O(|E(DFG)| \cdot |V(MRRG)|)$ constraints. Additionally, to guarantee no loops in the mapping, it requires that if two nodes are connected by an edge, at most one may have multiple fanins, resulting in a larger MRRG than absolutely necessary. While this approach generally results in a very large ILP, it is guaranteed to capture 100% of the flexibility of the architecture.

### III. Connectivity-Based CGRA Mapping

#### A. Connectivity

Consider the orthogonally-connected CGRA in Fig. 3a. A connection between adjacent processing elements is guaranteed, because of fully-connected input crossbars. 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). This is in contrast to typical FPGAs, where a less-than-fully populated input crossbar may mean that a connection to a neighbouring block is not guaranteed. With such certainty about data routing, taking the approach of the existing ILP provided with CGRA-ME [4] seems excessive. Especially on an architecture such as the one in Fig. 3a, it appears to be “overkill” to explicitly model all multiplexers and routing resources, and in this work, we also find this to be the case for the more complicated ADRES and HyCUBE architectures.

The ILP presented in this work (Section IV) is based around the higher-level principle that if an operation is mapped to a PE, then the fanin of the operation must be mapped to neighbouring PEs. The only ambiguous part of this being deciding exactly which PEs are neighbours.

Fortunately, CGRA-ME’s MRRG device model is a graph, making it amenable to graph-search-based methods. Unfortunately, this does not lead to a clear way of unambiguously...
determining which PEs are close together, i.e. which PEs are neighbours. A heuristic must be created, and one found to work well is a symmetric breadth-first search that is terminated after finding at least a certain number of PEs. After all neighbours have been found for each PE, an ILP can be constructed and solved.

The number of neighbouring PEs to identify before proceeding to 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 or torus connections. The number of neighbours that should be found in order to allow a DFG to be mapped is therefore architecture- and DFG-dependent. Most architectures will have a certain minimum “nearest-neighbour” number before any DFGs can be mapped, a certain point where more can be mapped, and a certain number where it is not fruitful to increase it any further. Effects of nearest-neighbour number choices 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 it may be beneficial to model routing congestion.

From a human view of the clustered architecture in Fig. 5, it is obvious that in the example CGRA there are limited connections between clusters, but an ILP that is only based on a set of neighbours for each PE is oblivious to this. In fact, if we do not model routing, we find that for architectures like these, we must iterate through many placements before finding one that can route. To include routing in the ILP, we follow two principles: 1) if a FU drives another FU, then a path from the fanin to the driven must be chosen; and 2) two paths that are driven by different FUs cannot share a vertex.

Sets of paths between FU nodes are found via an n-shortest-paths algorithm. A specific number of paths must be chosen, though 20 paths has been found to work well for all architectures tested in Section V. The potential downside of pre-computing paths and choosing from them is that some of the combinatorial expressiveness of a graph-based device model can be lost. Consider that between two PEs there may be a number of crossbars. The number of paths between these two PEs is at least equal to the product of the widths of the crossbars.

C. Composition

As a first pass, a reasonable approach is to choose a number of neighbours to search for, choose a number of paths to choose from between each FU, and then try to solve the resulting ILP. We found this to work well for very small architectures and benchmarks, but the extra variables required for choosing paths result in an ILP that is much larger, and does not scale well.

Contrarily, placement and “routing” can be completely split up, by not modelling paths at all to find a placement, and then testing if each placement can be routed by only choosing paths for the connections required by the placement. This approach has small ILPs for each stage, but the the placement ILP has no guidance as to routability, and consequently, it may not quickly find routable placements.

To solve this problem, one solution is to choose an ILP cost function that reflects how reliably a route can be found between two processing elements. Another solution is to add a form of congestion modelling. Instead of allowing at most one FU to drive a given non-FU vertex, this can be relaxed to allow up-to a small fixed number. And, to address the ILP size, a smaller number of paths are allowed. Between these two solutions, it was found that adding variables and constraints to allow taking the cost-based approach, the ILP size increased in a similar way, but actually was slower to discover routable solutions. Put another way, when looking for a routable solution, it was faster to give the ILP solver hard guidance, in the form of constraints, rather than soft guidance in the form of a cost function.

Both the cost- and constraint-based approaches produce big enough ILPs that the solver cannot consistently instantly (< 200ms) determine if it is infeasible. For example, the number-of-neighbours searched for may simply be too low to map...
the DFG, and the solver may take several tens of seconds to
prove that is infeasible. However, we also aim to keep the
number of neighbours as low as possible: a higher number
results in a ILP with larger constraints and more solutions –
many of which will be unroutable, e.g. requiring conflicting
route-through usage. To mitigate this, the original placement-
only approach proposed can be used as a test for a given
number of neighbours. Its small ILP model size makes it an
effective and quick solution, and we find that even though this
placement-only ILP does not model congestion, it is extremely
predictive of whether a solution exists to the larger placement-
with-congestion ILP.

The proposed algorithm can be summarized as the follow-
ing:

```python
for nn in CGRA.neighbourCountSchedule():
    if placeNoRoute(nn).failure():
        continue;
    for placement in placeWithCongestion(nn):
        routing = routePlacement(placement);
        if routing.success():
            return placement + 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 and ILP variables:

- **FU ⊆ V(MRRG)**: all MRRG nodes that can perform
  computation
- **comp(o ∈ V([DFG]))**: the set of MRRG FU nodes that
  o can be mapped to
- **neigh(u ∈ FU)**: a set of MRRG FU nodes that are
  reachable from u.
- **paths(a ∈ FU, v ∈ FU)**: a set of paths through the
  MRRG from u to v
- **f_{ou}**: is FU u used By DFG operation o?
- **e_{oupv}**: is there an DFG edge (o,p) mapped to FUs u and
  v?
- **p_{uwvi}**: is path number i used from FU u to v?

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

\[\forall u \in FU \sum_{o \in DFG} e_{ou} \leq 1\]

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.

3) **Fanin Required**: This constraint encodes the notion that
if an operation is mapped to a FU, then each DFG fanin of
the operation must be mapped to an neighbouring FU, i.e. that
\[f_{vp} \Rightarrow \exists u : e_{oupv}\] 
for each fanin o. This constraint ensures that all data that is needed by FU v will arrive, and is independent of how many places the needed data is computed, allowing for duplication/re-computation if the other constraints allow it. Other formulations are tend to require fanout instead of
fanin, and therefore do not allow re-computation.

\[\forall (o,p) \in DFG \forall v \in \text{comp}(p) \]
\[f_{vp} \leq \sum_{u \in \{x \in \text{comp}(o) : v \in \text{neigh}(x)\}} e_{oupv}\]

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. \(e_{oupv} \Rightarrow f_{ou}\).

\[\forall (o,p) \in DFG \forall u \in \text{comp}(o) \forall v \in \text{comp}(p) \cap \text{neigh}(u) \]
\[e_{oupv} \leq f_{ou}\]

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.
\(e_{oupv} \Rightarrow \exists q : p_{uvq}\).

\[\forall (o,p) \in DFG \forall u \in \text{comp}(o) \forall v \in \text{comp}(p) \cap \text{neigh}(u) \]
\[e_{oupv} \leq \sum_{q \in \text{paths}(u,v)} p_{uvq}\]
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, and even then they can only overlap initially. Situations other than initial overlap are resolved in a post-processing step that chooses the shorter of the two divergent path segments. Also, the constraint specified below will produce duplicate constraints, which are detected and not added to the ILP.

\[
\forall (o_1, p_1) \in \text{DFG} \forall q_1 \in \text{allpaths}(o_1, p_1) \forall (o_2, p_2) \in \text{DFG} \quad \text{p}_{q_1} + \sum_{q_2 \in \{\text{allpaths}(o_2, p_2) : \neg \text{comp}(q_1, q_2)\}} \text{p}_{q_2} \leq 1
\]

Where allpaths(o, p) is the set of paths between all comp(o) to all comp(p), i.e., \( \bigcup_{(u,v) \in \text{comp}(o) \times \text{comp}(p)} \text{paths}(u,v) \). And where \( \text{pcomp}(q_1, q_2) = (q_2 \cap q_1 = \emptyset) \lor (\text{driver}(q_2) = \text{driver}(q_1)) \), 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.

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

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 centre 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 estimated power consumption of using that path.

\[
\mathcal{L} = \sum_{o \in \text{DFG}} \sum_{u \in \text{comp}(o)} k_{ou} f_{ou} + \sum_{(o,p) \in \text{DFG}} \sum_{u \in \text{comp}(o)} \sum_{v \in \text{comp}(p)} l_{oupv} e_{oupv} + \sum_{u,v \in \text{FU}, q \in \text{paths}(u,v)} q^2 p_q
\]

V. EXPERIMENTAL RESULTS

A. Experimental Setup

Ranges of useful number-of-neighbours used for finding paths and constructing the ILPs were determined empirically, and presented in Table I. For a given architecture, the first number-of-neighbours tried is the “Min.” column, and this number is increased by “Step” until the value “Max.” has been tried. The number of paths used for the placement with congestion modelling is 3 for all architectures, and up to two paths may share a vertex. The number of paths used for the routing of a candidate placement is 20 for all architectures.

To simulate various sizes of architecture, and to demonstrate variability in runtime, we test each architecture for an II of 1,2 and 3 (i.e. 1, 2 and 3 CGRA contexts). Benchmarks used are the set of benchmarks used in [4] to characterize the existing CGRA-ME approach, plus some additional computation applications, and a fast-Fourier-transform benchmark from MiBench [13]. Each benchmark-architecture-II combination is run 6 times using different solver seeds, and with a limit of 7.5 hours – near the length of a typical weekday. We believe this represents how the existing approach might be used in practice: an architect might run several concurrent seeds to hedge against runtime variability.

The time taken to run the n-shortes-paths algorithm to find paths has been removed from every runtime of our approach by caching all data in memory before timing. 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 is the one provided by Gurobi Software [11], and experiments were performed using up to 4 threads on Intel® Xeon® Gold 6148 Processors.

B. Comparison & Discussion

Twenty-nine benchmarks over 3 architectures times 3 II values is too many data-points to include, so summaries of timing data collected are presented instead, in Tables II and III. We take geometric means across architecture-II and benchmark axes and compute the relative speedup. Maximum runtimes of this work are also presented. At the left of Table II, 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, for each architecture and II, the average runtime over 6 runs with different seeds is determined, and
the geometric mean of the averages is taken:

$$\forall b \in \text{benchmarks}, \quad \text{geomean}_{a,i \in \text{arches} \times \{1,2,3\}} \left( \frac{1}{6} \sum_{s \in \text{seeds}} \text{time}(b, a, i, s) \right)$$

The computation for column 4 is exactly the same, except runtime statistics 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 minimum number-of-neighbours required to map this benchmark on any of the architecture and II variants. Table III is similar, except the geometric means and maximums are taken across all benchmarks while fixing architecture and II. Also, II for each row is displayed in the second column.

Looking at Table II, 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 geomean runtime compared to some smaller, simpler DFGs (conv2, accumulate). This could be attributed to the solver quickly determining that FFT cannot fit in 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, while FFT simply maps in a reasonable amount of time on all architectures it fits on.

Looking at the maximum runtime for this work in Table II, we see that there is a stronger correlation between DFG size and maximum time taken, which is expected, as the placement-with-congestion ILP’s size strongly correlates with DFG size. However, looking at the minimum number of number-of-neighbours required to map the benchmarks reveals an even stronger correlation. As the number of path exclusivity constraints increases with the square of the number-of-neighbours, a benchmark requires a high number of fanout (exponential-$\ast$,cosh-4,cosh-4) or connections to distant FU nodes (FFT, long-chain) should take more time to map.

Table III shows average runtimes for each architecture/II variant across all benchmarks. Here, we observe a more straightforward pattern: an increasing II (which essentially acts as a multiplier on MRRG size) results in increasing runtime for the existing approach. 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 494 ×. The arithmetic mean of speedups is 5113 ×, and the median speedup is 1064 ×. The 90th percentile average runtime for the new approach is 7.9 seconds. If speedups where the existing approach times-out are dropped, then the speedup in geometric mean is 85 ×. For all mapping instances that the existing approach is able to provide a decision, the result of this approach matches.

Finally, some sample MRRG sizes, and corresponding ILP statistics are presented in Table IV. An II of 2 and the add-10’s DFG has 20 vertices – a middling value. The largest ILP used by this work when mapping benchmarks is the placement-with-congestion-modelling step. Numbers of constraints and variables are presented for this ILP adjacent to the same statistics for the single ILP in the existing approach. The placement-with-congestion ILP has approximately an order-of-magnitude fewer variables. Statistics for the placement-only and routing-only ILP of this work are also included underneath, and, with two orders-of-magnitude less than even placement-with-congestion ILP. Looking at the HyCUBE entry, the semi-independence from MRRG size can be seen, while constraint numbers for the existing approach follow an linear relationship with MRRG size. Models for mapping to HyCUBE consistently are significantly smaller than ADRES and Clustered, due to it requiring a consistently lower number-of-neighbours, which results in many fewer path variables and associated constraints.

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 & mapping decisions to the ILP/SAT solver, and modelling routing explicitly. All these approaches attempt to completely retain the flexibility of the architectures (or device model in the case of CGRA-ME) 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. Inspiration for the approach presented here comes from these works, and can be thought of as similar, but using an extremely 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 benefits of doing this 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 checking by trying to find a routing.

This approach is influenced by [9], [19]–[22], in that it primarily operates on connectivity information, but these other mapping procedures are each tuned/optimized for a specific class of architectures. The methods of [14], [18]–[20] manipulate the DFG in clever ways to assist in mapping the architectures (such as node duplication, and adding explicit “routing” nodes) but we cannot as easily apply this idea while remaining generic.

VII. CONCLUSION & FUTURE DIRECTIONS

We have presented an ILP based method for mapping applications to a variety of CGRAs in a reasonable amount

\footnote{[14] presents two approaches.}
### Table II
**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. | Min. NN Req. |
|----------------|----------|--------------|-----------|---------|----------------|--------------|
| cap            | 24       | .6           | .4        | 1.6     | 1.1            | 7            |
| sum            | 7        | 82.0         | .7        | 114.4   | 2.6            | 7            |
| nomen1         | 6        | 117.4        | .6        | 188.9   | 2.5            | 7            |
| add-10         | 20       | 322.5        | 1.0       | 129.1   | 7.3            | 9            |
| multiply-10    | 20       | 139.4        | 1.1       | 132.5   | 19.1           | 9            |
| weighted-sum   | 32       | 233.6        | 1.5       | 161.0   | 33.6           | 11           |
| FFT            | 38       | 242.1        | 9.6       | 25.2    | 478.7          | 12           |
| add-16         | 32       | 333.7        | 1.5       | 228.1   | 9.6            | 11           |
| multiply-16    | 32       | 350.7        | 1.5       | 239.9   | 8.6            | 11           |
| mac            | 11       | 401.4        | 1.0       | 418.8   | 3.5            | 7            |
| long-chain     | 35       | 477.0        | 3.1       | 154.5   | 77.0           | 11           |
| add-14         | 28       | 507.7        | 1.3       | 380.4   | 7.2            | 9            |
| multiply-14    | 28       | 508.2        | 1.4       | 352.9   | 6.7            | 9            |
| exponential-4  | 13       | 599.1        | .9        | 538.6   | 3.4            | 9            |
| taylor-series-4| 15       | 1,065.3      | .9        | 1,137.7 | 3.7            | 7            |
| exponential-6  | 26       | 1,270.5      | 7.9       | 161.5   | 61.6           | 14           |
| matrixmultiply | 17       | 1,371.0      | 1.5       | 944.3   | 5.7            | 7            |
| exponential-5  | 19       | 2,295.8      | 2.9       | 804.6   | 42.1           | 10           |
| cos-4          | 21       | 2,436.7      | 8.9       | 275.3   | 149.2          | 10           |
| cosh-4         | 21       | 2,449.8      | 7.9       | 309.2   | 133.4          | 10           |
| simple2        | 12       | 2,551.5      | 1.0       | 2,507.0 | 3.9            | 7            |
| simple         | 12       | 2,674.1      | 1.0       | 2,734.8 | 3.0            | 7            |
| conv2          | 16       | 3,091.3      | 1.2       | 2,474.1 | 4.3            | 7            |
| multisl-loop1  | 31       | 5,300.2      | 2.7       | 1,995.6 | 7.0            | 9            |
| conv3          | 24       | 5,469.7      | 2.0       | 2,690.8 | 4.7            | 9            |
| accum2         | 18       | 6,971.5      | 1.4       | 5,158.6 | 5.9            | 7            |
| accumulate      | 18      | 11,046.9     | 1.5       | 7,359.2 | 4.9            | 7            |
| mac2           | 24       | 15,372.4     | 2.2       | 7,095.2 | 8.5            | 9            |
| multisl2       | 25       | 19,866.9     | 2.6       | 7,594.1 | 9.1            | 9            |

### Table III
**Geometric Means of Six-Seed Benchmark Arithmetic Means, and Maximum Runtimes, by Architecture**

| Architecture Name | II | Existing [4] | This Work | Speedup | This Work Max. |
|-------------------|----|--------------|-----------|---------|----------------|
| adres 4x4         | 1  | 4.7          | .3        | 16.7    | 1.1            |
| clustered 2x2     | 1  | 9.7          | .4        | 22.5    | 1.0            |
| hycube 4x4        | 1  | 71.2         | .9        | 78.3    | 133.4          |
| adres 4x4         | 2  | 1,489.9      | 1.1       | 1,311.7 | 5.8            |
| hycube 4x4        | 2  | 6,462.3      | 4.6       | 1,414.8 | 603.6          |
| clustered 2x2     | 2  | 7,097.5      | 2.0       | 3,588.1 | 7.1            |
| adres 4x4         | 3  | 8,942.1      | 4.0       | 2,249.2 | 376.2          |
| clustered 2x2     | 3  | 10,095.4     | 5.2       | 1,924.0 | 427.0          |
| hycube 4x4        | 3  | 10,509.0     | 5.1       | 2,072.2 | 616.1          |

### Table IV
**Sample MRRG Sizes & Corresponding ILP Sizes for Add-10 Benchmark, with II = 2**

| No. Constraints | No. Variables |
|-----------------|---------------|
| Arch. MRRG Size | Existing This Work Existing This Work |
| ADRES 2320      | 53125 19148 104616 23896 |
| Placement       | 436   376   |
| Routing         | 163   390   |
| Clustered 4488  | 97813 17308 220024 21544 |
| Placement       | 484   424   |
| Routing         | 171   731   |
| HyCUBE 5056     | 115365 7468 220744 8952 |
| Placement       | 532   472   |
| Routing         | 179   419   |

of time, with significant improvement over the state-of-the-art in generic CGRA mapping [4], while minimally giving up mapping exactness.

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, and maybe take the “sliding window” approach of [16]. However, partitioning and incremental techniques are difficult to do generically, and must be done with care to ensure that the capabilities of the architecture being mapped to are taken into account. As discussed in Section V-B, reducing the number of path variables is essential to the performance of this work, so heuristic methods achieving this are of interest.
Computations were performed on the Niagara supercomputer at the SciNet HPC Consortium. [23] SciNet is funded by: the Canada Foundation for Innovation; the Government of Ontario; Ontario Research Fund – Research Excellence; and the University of Toronto.

ACKNOWLEDGEMENT

References

[1] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Harahazumi, and J. Anderson, “CGRA-ME: A unified framework for CGRA modelling and exploration”, in 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors, Jul. 2017, pp. 184–189. DOI: 10.1109/ASAP.2017.7995277.

[2] R. Hartenstein, “A decade of reconfigurable computing: A visionary retrospective”, in Proceedings of the Conference on Design, Automation and Test in Europe, IEEE Press, Munich, Germany, 2001, pp. 642–649. [Online]. Available: http://dl.acm.org/citation.cfm?id=367072.367839.

[3] R. Tessier, K. Pocek, and A. DeHon, “Reconfigurable computing architectures”, Proceedings of the IEEE, vol. 103, pp. 332–354, 2015. DOI: 10.1109/JPROC.2014.2386883.

[4] S. A. Chin and J. H. Anderson, “An architecture-agnostic integer linear programming approach to cgra mapping”, in Proceedings of the 55th Annual Design Automation Conference, ser. DAC’18, IEEE, New York, NY, USA: ACM, Jun. 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465799.

[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, ISSN: 1350-2387. DOI: 10.1049/ip-cdt:20030833.

[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 International Conference on Field Programmable Logic and Applications, P. Y. K. Cheung and G. A. Constantinides, Eds., Springer Berlin Heidelberg, 2003, pp. 61–70, ISBN: 978-3-540-45234-8.

[7] K. Patel, S. McG ettrick, and C. J. Bleakley, “SYSCORE: A coarse grained reconfigurable array architecture for low energy biosignal processing”, in IEEE International Symposium on Field-Programmable Custom Computing Machines, ser. FCCM ’11, IEEE, Washington, DC, USA, May 2011, pp. 109–112, ISBN: 978-0-7695-4301-7. DOI: 10.1109/FCCM.2011.38.

[8] M. Karunarathne, A. K. Mohite, T. Mitra, and L.-S. Peh, “Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect”, in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, Jun. 2017, pp. 1–6. DOI: 10.1145/3061639.3062262.

[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 Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, ser. PACT ’08, Toronto, Ontario, Canada: ACM, 2008, pp. 166–176, ISBN: 978-1-60558-282-5. DOI: 10.1145/1454115.1454140.

[10] L. Chen and T. Mitra, “Graph minor approach for application mapping on CGRAs”, ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 7, no. 3, 21:1–21:25, Oct. 2014. DOI: 10.1145/2655242.

[11] (). Gurobi optimization, [Online]. Available: http://www.gurobi.com/.

[12] S. Fortune, J. Hopcroft, and J. Wyllie, “The directed subgraph homeomorphism problem”, Theoretical Computer Science, vol. 10, no. 2, pp. 111–121, 1980. DOI: 10.1016/0304-3975(80)90009-2.

[13] 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 Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization, IEEE, Dec. 2001, pp. 3–14. DOI: 10.1109/WWC.2001.990739.

[14] G. Lee, K. Choi, and N. D. Dutt, “Mapping multi-domain applications onto coarse-grained reconfigurable architectures”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 5, pp. 637–650, May 2011, ISSN: 0278-0070. DOI: 10.1109/TCAD.2010.2098571.

[15] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaraningam, C. Estan, and B. Robatelli, “A general constraint-centric scheduling framework for spatial architectures”, ACM SIGPLAN Notices, vol. 48, no. 6, pp. 495–506, Jun. 2013, ISSN: 0362-1340. DOI: 10.1145/2499370.2462163.

[16] S. Chaudhuri and A. Hetzel, “Sat-based compilation to a non-venneumann processor”, in 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, Nov. 2017, pp. 675–682. DOI: 10.1109/ICCAD.2017.8203842.

[17] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, “SPR: An architecture-adaptive CGRA mapping tool”, in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA ’09, Monterey, California, USA: ACM, 2009, pp. 191–200, ISBN: 978-1-60558-410-2. DOI: 10.1145/1508128.1508158.

[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 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), IEEE, Jul. 2016, pp. 655–660. DOI: 10.1109/ISVLSI.2016.54.

[19] M. Hamzeh, A. Shrivastava, and S. Vrudhula, “Epimap: Using epimorphism to map applications on CGRAs,”
in Design Automation Conference 2012, Jul. 2012, pp. 1280–1287. DOI: 10.1145/2228360.2228600.

[20] ——, “Regimap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)”, in Proceedings of the 50th Annual Design Automation Conference, ser. DAC ’13, Austin, Texas: ACM, 2013, 18:1–18:10, ISBN: 978-1-4503-2071-9. DOI: 10.1145/2463209.2488756.

[21] W. Kim, D. Yoo, H. Park, and M. Ahn, “SCC based modulo scheduling for coarse-grained reconfigurable processors”, in 2012 International Conference on Field-Programmable Technology, Dec. 2012, pp. 321–328. DOI: 10.1109/FPT.2012.6412156.

[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 2010 Conference on Design and Architectures for Signal and Image Processing, IEEE, Oct. 2010, pp. 168–175. DOI: 10.1109/DASIP.2010.5706261.

[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, no. 1, p. 012026, 2010. DOI: 10.1088/1742-6596/256/1/012026.