MC-DeF: Creating Customized CGRAs for Dataflow Applications

GEORGE CHARITOPOULOS, School of Electrical and Computer Engineering, Technical University of Crete
DIONISIOS N. PNEVMATIKATOS, School of Electric and Computer Engineering, National Technical University of Athens
GEORGI GAYDADJIEV, Bernoulli Institute, University of Groningen and Department of Computing, Imperial College London

Executing complex scientific applications on Coarse-Grain Reconfigurable Arrays (**CGRAs**) promises improvements in execution time and/or energy consumption compared to optimized software implementations or even fully customized hardware solutions. Typical CGRA architectures contain of multiple instances of the same compute module that consist of simple and general hardware units such as ALUs, simple processors. However, generality in the cell contents, while convenient for serving a wide variety of applications, penalizes performance and energy efficiency. To that end, a few proposed CGRAs use custom logic tailored to a particular application's specific characteristics in the compute module. This approach, while much more efficient, restricts the versatility of the array. To date, versatility at hardware speeds is only supported with Field programmable gate arrays (FPGAs), that are reconfigurable at a very fine grain.

This work proposes MC-DeF, a novel Mixed-CGRA Definition Framework targeting a Mixed-CGRA architecture that leverages the advantages of CGRAs by utilizing a customized cell array, and those of FPGAs by incorporating a separate LUT array used for adaptability. The framework presented aims to develop a complete CGRA architecture. First, a cell structure and functionality definition phase creates highly customized application/domain specific CGRA cells. Then, mapping and routing phases define the CGRA connectivity and cell-LUT array transactions. Finally, an energy and area estimation phase presents the user with area occupancy and energy consumption estimations of the final design. MC-DeF uses novel algorithms and cost functions driven by user defined metrics, threshold values, and area/energy restrictions. The benefits of our framework, besides creating fast and efficient CGRA designs, include design space exploration capabilities offered to the user.

The validity of the presented framework is demonstrated by evaluating and creating CGRA designs of nine applications. Additionally, we provide comparisons of MC-DeF with state-of-the-art related works, and show that MC-DeF offers competitive performance (in terms of internal bandwidth and processing throughput) even compared against much larger designs, and requires fewer physical resources to achieve this level of performance. Finally, MC-DeF is able to better utilize the underlying FPGA fabric and achieves the best efficiency (measured in LUT/GOPs).

Authors' addresses: G. Charitopoulos, School of Electrical and Computer Engineering, Technical University of Crete, University Campus, Akrotiri Chania 73100, Greece; email: gcharitopoulos@isc.tuc.gr; D. N. Pnevmatikatos, School of Electric and Computer Engineering, National Technical University of Athens, Iroon Polytexneiou 9, Athens 15780, Greece; G. Gaydadjiev, University of Groningen, Nijenborgh 9 9747 AG Groningen.

1544-3566/2021/04-ART26
https://doi.org/10.1145/3447970

CCS Concepts: • **Computer systems organization → Data flow architectures**; • **Hardware → Reconfigurable logic and FPGAs**;

Additional Key Words and Phrases: CGRA, CGRA framework, reconfigurable computing, FPGA

ACM Reference format:
George Charitopoulos, Dionisios N. Pnevmatikatos, and Georgi Gaydadjiev. 2021. MC-DeF: Creating Customized CGRAs for Dataflow Applications. *ACM Trans. Archit. Code Optim.* 18, 3, Article 26 (April 2021), 25 pages.
https://doi.org/10.1145/3447970

1 INTRODUCTION

Specialized hardware accelerators of scientific applications are shown to achieve better performance and/or energy consumption [21, 27, 36, 38, 43]. However, designing and implementing accelerators is a difficult process requiring a combination of deep knowledge of the application, hardware description, or domain-specific programming languages and software tools. Throughout the years, several alternatives have been proposed in order to make this process easier. The dataflow paradigm is a promising and well-established approach toward the creation of customized hardware solutions. Several frameworks that extract dataflow graphs (DFGs) and map them on specialized hardware have been proposed [17, 37]. Still, mapping complex DFGs on FPGAs is a tedious process. The process is simplified using Coarse-Grain Architectures (CGAs), i.e., architectures that are used to map DFGs on pre-defined and fixed hardware. CGAs exploit hardware customization and often achieve faster and more energy-efficient execution. However, fixed/pre-defined hardware is also a drawback to these architectures as they lack in terms of flexibility and versatility compared to other solutions.

A more flexible solution compared to CGAs are *Coarse-Grain Reconfigurable Array (CGRA)* architectures, i.e., architectures that feature large reusable units with reconfigurable capacity [12], compared to more flexible approaches, which are highly customizable and tailored to the application's needs and requirements. The CGRA concept is a new incarnation of a decades-old research idea, which has been presented in research in various forms. Coarse-grain reconfiguration showed promise with successful projects such as RaPiD [13]. From a different perspective, the notion of large portions of an FPGA that are reconfigurable while the rest of the integrated circuit is in operation (i.e., dynamic partial reconfiguration) takes us back to at least the Xilinx Virtex II family from large volume FPGAs, and as far back as the Algotronix CAL series (a.k.a. Xilinx 6200) in the late 1980s and early 1990s. Nonetheless, there is a need to readdress this problem, because of the quantitative aspects of present-day FPGAs, applications, and CAD tools.

Since early reconfigurable architectures, e.g., programmable logic arrays (PLAs), the basis of mapping has been to map in a *1-to-1* fashion a single application gate/function onto a single gate of the architecture. CGRA application mapping, although improved, continues to follow this paradigm limiting the capabilities of CGRAs. Additionally, despite their massive customization abilities, CGRAs still strive to achieve efficient mapping while being as generic as possible. So far, few related works in the field of CGRAs have stirred away from a *1-to-1* mapping methodology, i.e., mapping one compute function/operation on a single CGRA cell. This intentional lack of multiple-instruction–multiple-data cell operation leads to sub-optimal designs in terms of offered acceleration and general execution time. Therefore, we need a CGRA definition framework that is able to (a) map multiple nodes in one cell and (b) offer customized cell functionality, while maintaining a degree of flexibility.

A Mixed-CGRA architecture combines advantages from both the CGRA and the FPGA paradigms, using both a coarse-grain cell structure and an amount of (LUT-based) reconfigurable logic, for added flexibility, connected with a fast and high-bandwidth communication infrastructure. In this article, we present holistically, a *Mixed-CGRA Definition Framework (MC-DeF)*, which uses the application's DFG representation and performs application analysis in order to decide the CGRA-cell structure and functionality, mapping and routing of the target application on the resulting architecture, and finally, area occupancy and energy consumption value estimations based on the created design. Implementation options of the resulting design are as an ASIC circuit, with added flexibility through the use of the adjacent reconfigurable LUT array, or as an overlay to an FPGA.

A key advantage of MC-DeF is a new degree of abstraction, when considering application functions/gates/nodes, that allows for application-customized CGRAs that are also able to accommodate applications with similar size and computational characteristics. We introduce a mapping algorithm that allows for efficient mapping of multiple application nodes in one cell, i.e., many-to-one mapping. Finally, the inclusion of a few key but easy-to-understand threshold values, cost functions, and restrictions in the MC-DeF workflow allows the user to create design tailored execution, area, or energy requirements. This article expands on our previous works [5, 6] and proposes a complete and novel *MC-DeF* coupled with a *Mixed-Coarse-Grain Reconfigurable Array (Mixed-CGRA)* architecture. In this article, we introduce three new features in MC-DeF's toolchain:

— An MC-DeF extension to map applications on a CGRA customized for a different application.
— An MC-DeF extension to support CGRA cells that include a small amount of reconfigurable logic for additional flexibility.
— A novel Mapping Improvement Algorithm used in the mapping process of MC-DeF.

Additionally, we perform several experiments using nine scientific applications in order to

— evaluate MC-DeF as a holistic framework and the resulting designs of the nine applications;
— evaluate the advantages in performance when we introduce reconfigurability within the CGRA cell;
— showcase the ability of MC-DeF to perform flexible mapping of new applications on existing CGRAs without loss of performance;
— evaluate the novel Mapping Improvement Algorithm; and
— compare our framework with state-of-the-art related works in the field of CGRA architectures.

The rest of the article is structured as follows: Section 2 presents related work on the field of CGRA architectures, while Section 3 presents an example of the problem we want to solve with MC-DeF. MC-DeF and the targeted Mixed-CGRA are described in Section 4. Evaluations of MC-DeF and comparisons with other works are in Section 5. Finally, Section 6 concludes our work and presents our final remarks.

2 RELATED WORK

Our main focus on related work in the field of CGRA architectures is architectures that have been evaluated using FPGA devices or act as FPGA overlays. Stojilovic et al. present a technique to automatically generate a domain-specific coarse-grained array from a set of representative applications [40]. Their technique creates a shortest common super-sequence found among all input applications based on weighted majority merge heuristic. Using this super-sequence, the framework creates a cell array able to map the application's instructions.

Several works attempted to combine the DFG paradigm with CGRAs [31, 34, 45]. Most of them focus on domain-specific architectures based on statically defined functional units (FUs). This approach limits the abilities regarding the contents of the CGRA "cell."

REDIFINE [2] is a polymorphic ASIC in which specialized hardware units are replaced with basic hardware units that can create the same functionality by runtime re-composition. The high-level compiler invoked creates substructures containing sets of compute elements. Paired with REDIFINE, HyperCell [33] enhances the CGRA compute elements with reconfigurable macro datapaths that enable exploitation of fine grain and pipeline parallelism at the level of basic instructions in static dataflow order.

Mapping dataflow applications on CGRAs is a wide research field. Niedermeier et al. present a novel programming paradigm designed to combine the principles of dataflow execution with CGRAs [34]. The authors present a Haskell-based programming language coupled with a CGRA architecture comprising reconfigurable cores. Each core includes a FU, a register file, and a program memory.

Intermediate Fabrics (IFs) is an overlay architecture consisting of 196 heterogeneous FUs with an island-style interconnect [10, 28]. The complete CGRA is implemented on an Altera Stratix III FPGA in order to support fully parallel, pipelined implementations of a set of image processing kernels. The DySER architecture consists of a heterogeneous array of 64 FUs interconnected with a programmable network [15, 16, 23]. A key disadvantage of DySER is high LUT consumption. Early implementations were only able to fit a 2×2 32-bit DySER on the FPGA. Subsequent implementations used DSP blocks as the homogeneous FU, thus achieving larger arrays.

FPCA, fully pipelined and dynamically composable architecture [9], was proposed as an array of clusters, organized in a mesh with Nearest-Neighbor style interconnect, where each cluster consists of a set of Processing Elements (PEs). PEs are connected by a permutation network with a high connectivity within a cluster, and then by a global NN style interconnect for more scalable connectivity. The Soft Coarse-Grained Reconfigurable Array (SCGRA) overlay [30] was proposed to address the FPGA design productivity issue, demonstrating a 10−100× reduction in compilation time compared to the AutoESL HLS tool. Application-specific SCGRA overlays were subsequently implemented on the Xilinx Zynq platform [29], achieving a speedup of up to 9× compared to the same application running on the Zynq ARM processor.

In [32], the authors using various configurations of tile-array overlays create just-in-time accelerators. They identify several functionally independent programming patterns of an application, implement them, and place them on partially reconfigurable tiles. Results show a considerable increase in resource utilization, while recording similar or better acceleration rates, when compared to a standard High-Level Synthesis (HLS) implementation of the complete application.

Apart from using heterogeneous customized FUs, several researchers have elaborated on the use of digital signal processing (DSP) blocks as the CGRA FU. A fully pipelined DSP block-based overlay architecture is presented in [22]. The overlay uses the dynamic programmability of the DSP block and maps up to three operations to each node (one add/sub, one mul, one ALU op), resulting in a significant reduction in the number of processing nodes required. DeCO [22, 24], uses the same principle as [22] but the CGRA is arranged in a cone-shaped cluster of DSP-based FUs utilizing a simple linear interconnect between them.

Another type of CGRA architecture is the expression-grained reconfigurable array (EGRA) [3]. The architecture is described as a template with each PE hardware able to be customized prior to fabrication. The authors, by analyzing patterns in the application's computations, decide on a set of arithmetical/logical units to implement in the PEs.

Table 1. CGRA Frameworks Features Checklist

Features	Related Work						
	EGRA [3]	DySER [15, 16]	IF, IF opt. [10]	Stitch [41]	Stojilovic Domain Specific CGRAs [40]	DECO [22, 24]	MC-DeF
Application operation/function -based CGRA cell	✓	✓	✓	✓	✗	✓	✓
Many-to-one mapping	✓	✗	✗	✓	✓	✗	✓
Commonly used function chain discovery	✓	✗	✗	✓	✓	✓	✓
LUT array flexibility	✗	✗	✗	✗	✗	✗	✓
One CGRA many applications	✗	✓	✓	✗	✓	✗	✓

Stitch [41] is a many-core architecture with tiny, heterogeneous and configurable accelerators inserted at each core, designed for smart-wearable devices. The accompanying compiler is able to combine smaller accelerators, or parts of them, i.e., *patches*, from different cores to form large and complex ones. *Patches* from different cores communicate via a dedicated single-cycle multi-hop SMART NOC.

The CGRA-ME framework [7] starting from an arbitrary CGRA description and a benchmark application creates (a) a simulated design for logic functionality verification, (b) a configuration bitstream for the CGRA, and (c) a Verilog RTL architecture model of the final design. Compared to MC-DeF, this work omits the cell definition phase that creates application-based customized designs. Additionally, it uses a high-cost Simulated Annealing Mapper to map and route the CGRA design, while MC-DeF uses cost-function–based *Node Mapping* technique for mapping and a lightweight cost-function–based *Mapping Improvement* algorithm for routing.

While MC-DeF bares a resemblance to many of the works stated in this section, it stands out as being one of the few works adding LUT array-based versatility as well as creating unique CGRA cells that can be either domain- or application-specific depending on the user's requirements. Contrary to [40], MC-DeF employs a technique that tries to find common sequences of operations within one application. This leads to a more application-specific CGRA with the flexibility added through the use of an adjacent LUT array. MC-DeF and EGRA [3] are the only frameworks able to create an operation/function-based CGRA cell based on application analysis. While similar in principle, MC-DeF performs a more detailed application analysis, including the cell chains of commonly used functions instead of just the stand-alone functions. Table 1 offers a checklist of features used in other related works and in MC-DeF. We can observe that the LUT array flexibility feature is only available in MC-DeF. Additionally, our framework is the only one able to provide the user with all the key features listed here. In Section 5, we evaluate our approach and compare it to *Intermediate Fabrics, Intermediate Fabrics (opt), DySER, and DECO.*

3 CREATING CGRAS FROM DFGS

The problem we address in our work is the following: Given an application dataflow graph, find a suitable set of nodes to implement in a CGRA cell, map the graph nodes unto the cells created in the previous step, and place and route the resulting transformed application DFG in the CGRA architecture shown in Figure 1. The figure depicts the cell array (CE grid), the connectivity network, and the adjacent LUT array that enables mapping arbitrary functions that are not very common in the application. The picture also depicts the internal structure of the cell with the network configuration memory (CM), the implemented application nodes (APP_N), and the necessary FIFOs.

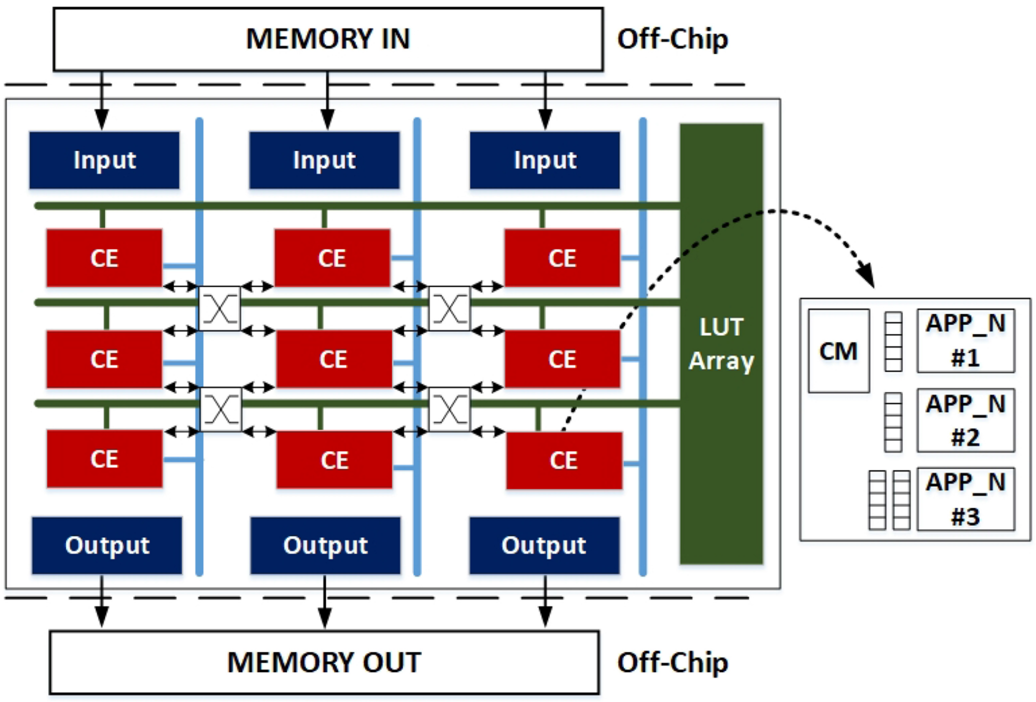


Fig. 1. Structure of the proposed CGRA.

The whole process performed by our framework is shown in Figure 2. The framework consists of four phases:

- **Cell Structure and Functionality:** the process of determining the structure and functionality of the CGRA cell(s).
- **Mapping:** the process of mapping the application's computational elements on the CGRA cells and the accompanying LUT array.
- **Routing:** the process of connecting the different cells using switch boxes and the underlying network.
- **Area and Energy Estimation:** the process of estimating the occupied area and the energy consumption of the resulting chip.

Our intention is for MC-DeF to be a self-contained tool able to aid the user in creating customizable CGRA designs from an application's DFG. To obtain that, the user needs to first invoke a High-Level Synthesis tool or another compilation-style front-end. As a front-end for our framework, we use the Maxeler Platform, which uses the Dataflow Paradigm to enable massive amounts of hardware acceleration. Maxeler Technologies is an HPC company that specializes in Multiscale Dataflow Computing (MDC) [35].

The Maxeler framework uses as input a high-level language implementation of a dataflow application and after synthesis creates a dataflow graph. The DFG created contains a high abstraction hardware representation of the input applications. The nodes of the DFG are hardware modules such as adders, counters, multipliers, and so forth. The DFG nodes are an intermediate representation between the high-level code and the resulting hardware.

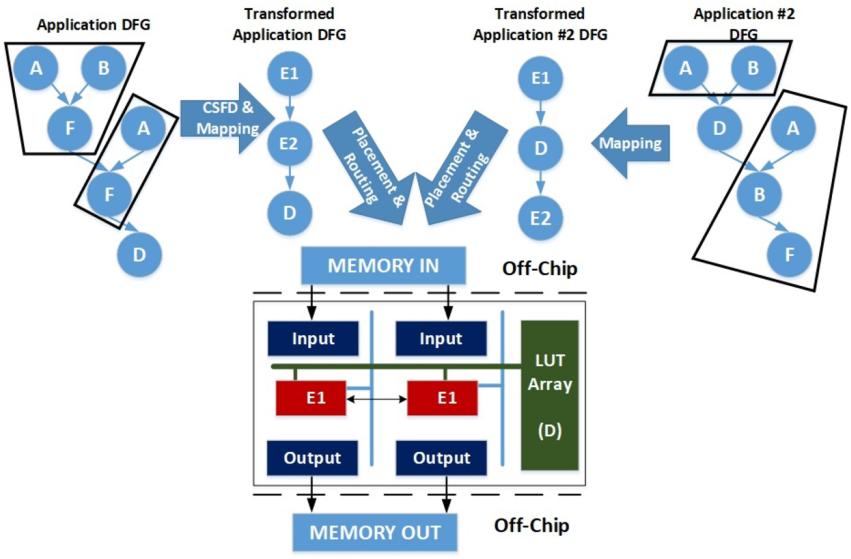


Fig. 2. An example execution of MC-DeF for two different application DFGs.

MC-DeF creates a fully routed design based on a user-provided architectural template. The architecture proposed here tries to balance the versatility and generality aspects of the framework. The basis of the architecture is a cell array, responsible for the computational needs of the mapped application. Additionally, we include LUT-based logic to support versatility of our architecture. The LUT-based logic can be integrated to our architecture either as a separate and adjacent LUT array or as LUT logic included in the CGRA cells.

The CGRA architecture proposed in this article uses a two-level network as its communication infrastructure. As is common with most CGRA architectures, the communication infrastructure is divided into local and semi-local/global. The first local level serves cell-to-cell communication. The implemented network for cell communication is a modified DMesh network [19], a mesh-type network-on-chip (NOC) architecture, which enhances the typical 2D-Mesh network with diagonal connections. DMesh is able to improve average latency and saturation traffic load; the authors report on a 25% reduction of inter-node distance over a classic 2D-Mesh, while using almost double the resources. In our MC-DeF defined architecture, we use a modified DMesh network that implements only bi-directional diagonal connectivity. The rationale behind this choice is the typical structure of a dataflow graph with input data coming from the top of the graph (input) and flowing downward (toward the outputs). This idea is further exploited in the DECO architecture [24] with the cone-shaped array structure.

The elimination of four connection points does not affect our design since their use is to alleviate network congestion. In our case, this is not necessary since very few cell-to-cell connections are bi-directional, and also all computations are pipelined. Additionally, we reduce the resource utilization of the array at the expense of additional distance between same row/column cells. Same row/column connectivity is implemented with the inclusion of a bi-directional crossbar switch at the intersection of the diagonal links and a bypass logic inside the cell's connectivity matrix.

The second level of the MC-DeF network is a grid of wide vertical and horizontal bi-directional buses independent to each other. The light-blue lines in Figure 1 are used for data input/output

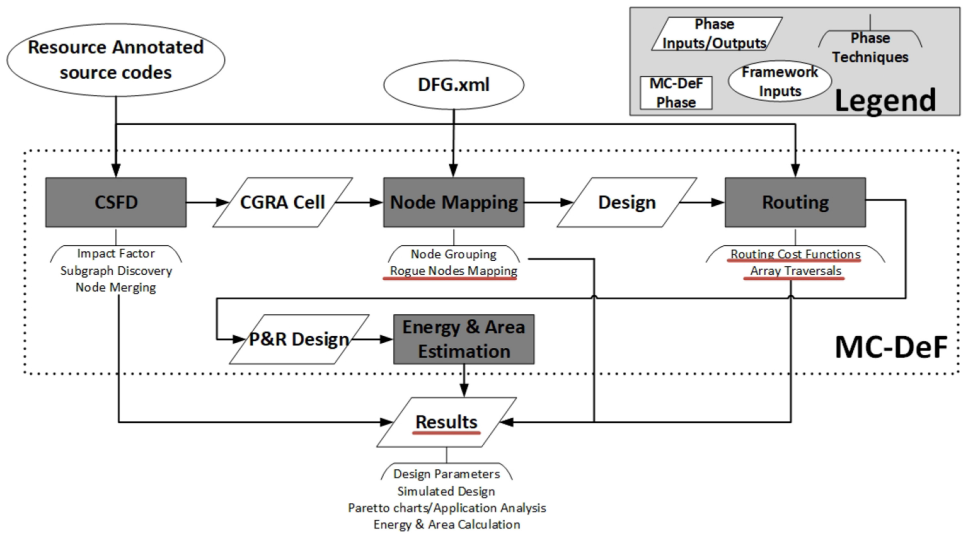


Fig. 3. The MC-DeF flow with its associated phases and techniques.

movements to/from our array. The number of buses used is relative to the number of cells on the x-axis of our cell array. Each cell has a bi-directional connection with the mentioned bus. While it is common for input/output connections to be directly connected with edge-placed cells, in our case we opt toward a fully connected grid, that will enable flexibility in our placement algorithms in the mapping phase of MC-DeF. Also, we use horizontal bi-directional buses, green lines in Figure 1, to establish communication between the cell and LUT arrays. The number of buses used is relative to the number of cells on the y-axis of our cell array.

If the user opts for a LUTs-in-cell approach a third-level, i.e., the LUT network, is added to the communication infrastructure responsible for transferring data generated by LUT-based nodes. The third level works in parallel and transparently from the other two, which further increases the internal bandwidth of the design. However, the user has to consider the logic overhead required to implement the third level of the network.

Finally, to increase versatility, we opt toward the ability of our framework to map additional applications on an already configured CGRA design, shown on the right-hand side of Figure 2. The main difference between the two MC-DeF runs is that now we skip the Cell Structure and Functionality Definition phase. The mapping and routing phases are carried out based on the cell structure created during our first MC-DeF run.

4 MIXED-CGRA DEFINITION FRAMEWORK

This section presents the Mixed-CGRA Definition Framework and its associated processes and algorithms, highlighting the extensions compared to previous work [5, 6]. The complete process carried out by MC-DeF is shown in Figure 3. The gray rectangles denote the phases of MC-DeF: (a) Cell Structure and Functionality Definition (CSFD) phase (Section 4.1) decides on the contents of the CGRA cell, (b) the Node mapping phase (Section 4.2) creates a modified DFG graph using the resulting CGRA cell, (c) the Routing phase (Section 4.3) creates a mapped and routed design ready to place on an FPGA device, and (d) Energy and Area Estimation phase (Section 4.4) presents the user with estimates of the created CGRA design.

4.1 CSFD Phase

The majority of current CGRA architectures use a static and pre-defined set of compute elements, soft-core processors, and/or ALU elements coupled with instruction memories to create a compute cell. While these kinds of approaches have proven highly flexible and are general enough to map a wide range of applications, they lack in (a) application scaling, (b) resource utilization, and (c) total number of operations performed in parallel. With MC-DeF, we opt toward a CGRA cell able to perform more than one operation in parallel, includes high abstraction hardware modules, and is able to implement a wide range of applications through cell-network communication reconfiguration.

To create highly customized cells optimized to the target application's characteristics, MC-DeF utilizes three techniques. The first one is the *Impact Factor* metric, introduced in [5], which denotes the estimated resource impact a DFG node has on the actual resource utilization of the application, i.e., the percentage of LUTs, FIFOs, BRAMs, and DSPs used by a node, over the total resource usage of the application. Nodes with high *Impact Factor* are labeled for inclusion in the cell structure.

The second technique is Frequent Sub-Graphs Discovery, a process bearing a strong similarity to the one used to identify frequent chains of instructions [8]. In [5], the authors run a modified version of GraMi [14], an algorithm for extracting frequent sub-graphs from a single large graph. A graph is extracted only if it exceeds a frequency and a resource utilization threshold, thus limiting the search space to sub-graphs that have high occurrence frequency and use the most hardware resources.

The third technique deals with a critical issue in CGRA design: oftentimes nodes have the same sequence of operations but apply it on different bit-widths. The naive approach considers these nodes as separate, leading to CGRA designs that are harder to route due to cell heterogeneity. To address this, we include in the CSFD phase *Node Merging*; an algorithm designed to find whether two nodes with the same functionality should be merged under the same bit-width and what the optimal bit-width for the current application is, described in detail in [6]. We use two metrics for Node Merging: the bit-width difference between the two nodes, and the *Percentage Gain* of merging these two nodes.

Through the *CSFD* phase, MC-DeF decides which DFG nodes will be implemented within the CGRA cell and ensures that the functionality of the CGRA cell is beneficial in terms of resources, frequency of occurrence in the DFG, and the bandwidth achieved among cell communication. The threshold values applied are subject to change according to user needs and design restrictions.

4.2 Node Mapping

Node mapping is the process of assigning DFG nodes of an application in CGRA cells. However, CGRA cells may contain multiple independent or chained functions, making the problem of mapping nodes to cells a difficult algorithmic process. In order to efficiently allocate the DFG nodes on the CGRA cells, we implement a novel *Node Mapping* algorithm. Starting from an application DFG using nodes $N = A, B, C, D$, MC-DeF decides on the contents of the CGRA cell on the CSFD phase; the resulting CGRA cell contains one stand-alone node and a sub-graph (right arrow notation denotes a sub-graph inclusion in the CGRA cell), i.e., $E = A \rightarrow C, B$. In the mapping phase of MC-DeF, we want to create a new graph that emits the nodes included in the CGRA cell and substitutes them with a new node, *Node E*, which describes the resulting CGRA cell as shown in Figure 4. Ultimately, the hardware elements included in E1 and E2 are the same but their utilization differs; as can be seen, only the sub-graph $A \rightarrow C$ is used in E2. *Node Mapping* finds and evaluates possible coverings/mappings of the DFG using two cost functions:

- Unutilized cell resources: this cost function measures the amount of unused resources among all the CGRA cells. A CGRA cell consisting of three nodes, with only two of them used, will have an unutilized cell resources count equal to one.
- Connections between cells: this cost function measures wire connections between different CGRA cells.

The pseudo-code for the node mapping algorithm used in MC-DeF is shown in Algorithm 1. The mapping algorithm considers all the nodes or chains of nodes implemented in the customized CGRA cell. If the CGRA cell contains a sub-graph, i.e., two or more nodes explicitly connected, the algorithm finds all the corresponding nodes (nodes between *node_source_pattern* and *node_dest_pattern*) and places them in a cell structure, with the *insert* function. Then, for each

ALGORITHM 1: Node Mapping

```
Result: Best mapping of DFG nodes in cells
inputs = master_cell, DFG_Nodes, DFG_Edges;
while runs == 100 do
    mapped_nodes = 0;
    for item in master_cell do
        /* For each pattern or individual node in master_cell structure find all the corresponding
           instances in the DFG and place them in cells.                                          */
        if pattern in master_cell then
            for edge in DFG_Edges do
                if edge in pattern then
                    cell =insert(node_source_pattern);
                    cell =insert(node_dest_pattern);
                    extra_nodes = search_adjacent_nodes(cell, DFG_Nodes);
                    /* Removes nodes based on cell utilization and                                */
                    /* distance of extra nodes in each possible mapping                           */
                    check_with_other_cells(Cell_list, C, extra_nodes);
                    cell = insert(extra_nodes);
            end
        else if No patterns in master_cell then
            for edge in DFG_Edges do
                if edge part of master_cell then
                    cell = insert(node);
                    mapped_nodes = mapped_nodes + 1;
                    /* At this point extra nodes include non-connected pattern nodes              */
                    extra_nodes = search_adjacent_nodes(Cell, DFG);
                    check_with_other_cells(Cell_list, C, extra_nodes);
                    cell = insert(extra_nodes);
            end
        mapping = insert(cell);
    end
    cell_util = measure_cell_utilization(mapping);
    cell_con = measure_cell_connectivity(mapping);
    best_mapping = compare_mappings(mapping, best_mapping);
    run++
end
non_mapped_nodes = search_non_mapped(DFG_Nodes, best_mapping);
```

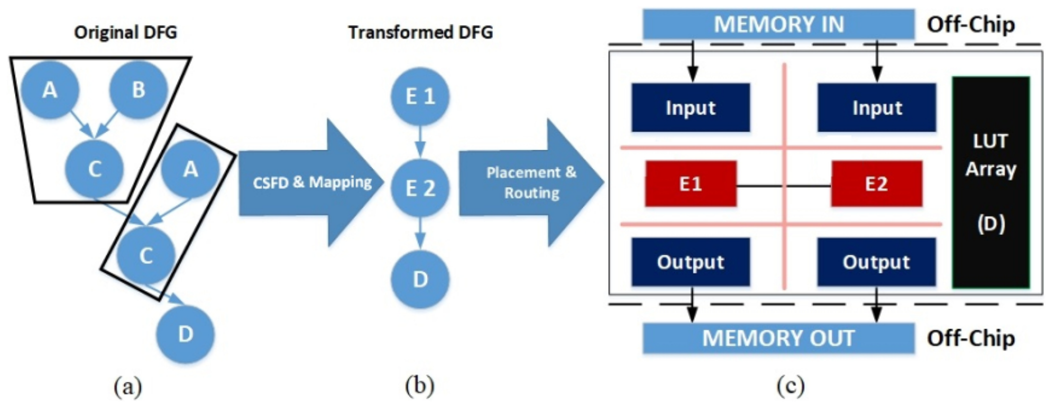


Fig. 4. Node Mapping in the MC-DeF. (1) CSFD phase decides on the CGRA cell contents; (2) Mapping creates a DFG using the new node E, the functionality of which is equivalent to the CGRA cell defined; (3) and (4) Placement and Routing phases place the nodes in a CGRA grid and decide the connectivity between them.

of the nodes already placed in the cell, through the use of the *search_adjacent_nodes*, the algorithms record all the DFG nodes that are adjacent, i.e., with a direct connection, or close, with minimum distance, to the ones already in the cell, and stores them in the *extra_nodes* structure. These nodes are then being checked against all other placed nodes in the cell array, with the *check_with_other_cells* function, and the ones already placed are removed from the *extra_nodes* structure. The remaining nodes are all inserted in the cell provided adequate available logic, or, if not, the algorithm chooses at random which ones to insert in the current CGRA cell. At this stage non-inserted nodes are stored and prioritized for inclusion in subsequent runs of the mapping algorithm. The same process is repeated if the current processed CGRA cell logic is not a chain of nodes. For each unique mapping created, the algorithm measures the *Unutilized cell resources* and the *Connections between cells* cost functions and chooses the mapping that best minimizes them. This process is repeated for 100 mappings. This number is a design time parameter that can be fixed accordingly by the user depending on the required effort spent by the framework to find an optimal solution. Finally, with the use of the *search_non_mapped* function, the mapping process records all the nodes not able to be placed within a cell; these nodes will later be placed in the LUT structure available.

Even though some nodes are not directly mapped to CGRA cells, e.g., node D in Figure 4, *CSFD* phase strives to ensure that these nodes are but a small fraction of the total resources used by the application. However, it is necessary to map these nodes in the Mixed-CGRA design. MC-DeF offers the user two alternatives for *"rogue" nodes.*

- **LUT array:** An adjacent LUT array able to accommodate all the DFG nodes not directly mapped to the cell logic.
- **LUTs-in-cell:** The remaining *rogue nodes* are implemented in small LUT array structures placed inside the CGRA cells.

The **LUT-array** approach is straightforward in terms of implementation from MC-DeF. First, during the *Node Mapping* phase, any node that is not included in the cell array is labeled for LUT-based implementation on the LUT array. Then, the *Routing* phase establishes the grid network responsible for transferring data to and from the LUT array. Node implementation and mapping within the LUT-array structure is similar to mainstream FPGAs. The LUT array is not treated as a CE because the intention is to offer a degree of reconfigurability that CGRA cells do not.

The **LUTs-in-cell** (*L-i-C*) option is more complex. First, we have to take into consideration the size of the individual LUT structures and to keep an almost uniform distribution among the CGRA cells. The inspiration for this idea was the Stitch architecture [41] and its ability to create heterogeneous and configurable core tiles. Additionally, we ideally want to place *rogue nodes* inside cells that have a direct connection with, e.g., RN 1 takes inputs and gives output to nodes placed in Cell 2, so we intuitively want to place it in the cell's 2 LUT structure.

For more complex decisions, we invoke the cost functions implemented in the *Routing* phase of MC-DeF and make placement decisions accordingly. The routing cost functions are taken into consideration because for *rogue nodes* there are no resource-related restrictions. For routing purposes a separate network is implemented working transparently from the cell network. The two networks communicate via dedicated buffers.

The algorithm tries to find a mapping that minimizes the cost functions and terminates its execution after discovering at least 100 mappings. Each mapping is different depending on which of the adjacent nodes will be selected for cell inclusion. The above number is used empirically through experimentation with the applications used to evaluate MC-DeF. Further increase of the number of minimum mappings discovered could yield better overall mapping results but at the cost of increased execution time. This number can be tailored by the end user of the framework.

4.3 Cell Routing

In this phase, MC-DeF establishes the connections needed to route the cell array, as well as the input/output infrastructure of the design. The *Routing* phase of MC-DeF uses two cost functions in order to create designs with low communication overhead: the number and size of synchronization FIFOs used in each cell and the distance of two communicating cells.

Dataflow execution dictates that operands arrive at compute nodes synchronized. However, operands from different paths may observe different compute and communication latencies. MC-DeF uses synchronization FIFOs where needed to re-time inputs in each CGRA cell. Synchronizing cell-node inputs could be remedied—but not fully solved—by latency-aware mapping of the cells; however, this would lead to increasing the overall latency of all the cell array. By inserting synchronization FIFOs inside the cells, we ensure unobstructed parallel and pipelined execution.

Cells are recognized by their position in the array, i.e., vertical and horizontal coordinates. For two cells that exchange data between them, their distance is equal to the number of D-Mesh bi-directional crossbar switches between them. For example, the distance of cell A (0,0) and cell B (2,1) is 2. After calculating the cell distance between two connecting cells, the synchronization FIFOs are formulated accordingly. The distance between cells and, Input/Output nodes and the LUT array is three since communication is achieved over the slower grid network. The distance between the nodes within the LUT array is not considered.

These cost functions are used for improving the communication infrastructure. The next step of the routing process is to minimize them using a Mapping Improvement algorithm. Through multiple trials, we observed that simultaneously minimizing both metrics is not possible. Instead, we focused the minimization on the metric with the largest variance among its values. As a result, the Mapping Improvement Algorithm focuses on minimizing the distance of two communicating cells.

For the two cells mentioned before, we move one of them along the axis that shows the largest distance. For example, moving Cell A to the (1, 0) position reduces the distance by 1. After this cell movement we need to re-calculate the average distance per cell compared with the previous value and perform more cell movements if necessary. The process is repeated until a local minimum

Table 2. Energy Consumption of Electronic Circuits Used in MC-DeF

Circuit (32-bit double percision)	Energy (pJ)
Node Add/Sub	10
Node Multiply	9
Node Division	13
Logic Gates Nodes	0.5
64-bit read from an 8-KB SRAM	2.4
Data movement between cells	0.115 pJ/bit/mm

Table 3. Area Occupancy Estimations of Electronic Circuits Used in MC-DeF

Circuit		Area (μm^2)
Node Add/Sub Node Multiply/Divide		2,500
FIFO (bits)		
Width	Depth	
<=8	<=1,000	250
>8	<=1,000	250
<=8	>1,000	$\lceil Depth/1,000 \rceil$ * 250
>8	>1,000	$\lceil Depth/1,000 \rceil$ * $\lceil Width/8 \rceil$ * 250

value is found, after a finite number of movements. In Section 5, we provide results on the number of movements required to reach a local minimum for our example applications and the percentage of improvement achieved by our Mapping Improvement Algorithm.

4.4 Area and Energy Estimations

The overall cost of the resulting CGRA architecture is evaluated by measuring the area of the resulting architecture and the energy consumption. Similar to other state-of-the-art related works ([4] and [39]), we estimate the area occupancy of our architecture assuming a 7 nm lithography technology. Thus, a 6 T SRAM bit cell unit's size is 30 nm^2, i.e., 38.5 Mb in 1 mm^2. For example, a 1k $\times$ 8-bit FIFO will occupy approximately 250 μm^2, while the area needed to implement a fused double precision Multiply-Accumulate on 7 nm is 0.0025 mm^2. Additionally, we consider two 19.5-mm^2 Input/Output infrastructures at the top and bottom of the CGRA with 13 mm length and 1.5 mm width. Also, the LUT array area is calculated based on [1, 44]. The numbers reported by the area evaluation phase of MC-DeF are *CGRA-only*, *CGRA+I/O*, and *Total (CGRA+I/O+LUT) Area* in mm^2.

Calculating energy consumption of the resulting Mixed-CGRA design is based on the individual computing elements used. Dally in [11] shows how the 64-bit double precision operation energy halved from 22 nm to 10 nm. Additionally, in [26] Dally and co-workers accurately measure the energy consumption of several electronic circuits on a 10 nm lithography technology. The numbers reported in this study are the basis of our energy consumption estimations and constitute a pessimistic estimate for a 7 nm lithography.

In Tables 2 and 3, we present the area and energy estimations considered by our MC-DeF framework. The nodes presented in these tables are the ones found in the application DFGs used for our studies and initial calibration of the MC-DeF. The system interconnect access requires 1000 pj.

Table 4. MC-DeF Metrics, Thresholds, and Cost Functions

Name	**Type**	**MC-DeF Phase**
Impact Factor†	*metric*	CSFD Application Analysis
Utilization of frequently occurring sub-graphs†	*threshold*	CSFD Sub-graph Discovery
Frequency of frequently occurring sub-graphs†	*threshold*	CSFD Sub-graph Discovery
Percentage Gain†	*metric* *(threshold applied)*	CSFD Node Merging
Bit-difference†	*metric* *(threshold applied)*	CSFD Node Merging
Connections between Cells*	*cost function*	Mapping
Unutilized Cell Resources†	*cost function*	Mapping
Cell Distance*	*cost function*	Routing
Number and Size of Sync. FIFOs*	*cost function*	Routing

Entries annotated with ∗ are used for Communication infrastructure optimization, and those with † for CGRA array optimization.

Additionally, in the MC-DeF energy and area consumption estimations, we assume 100% utilization of the cell and LUT arrays on a fully utilized pipeline dataflow path. These values are worst-case scenarios, so they correspond to highly overestimated scenarios. Additional optimizations at the implementation level would allow for more efficient designs.

4.5 Discussion

The Mixed-CGRA reconfigurable designs produced by MC-DeF are technology agnostic. Two main avenues for the implementation of these designs are (i) on FPGAs, and (b) as custom ASICs. The former option is typical in the CGRA research field, and we can take advantage of the FPGA reprogramming and use MC-DeF results as an overlay structure. The overlay, together with the data transfer protocol and framework forms a complete system. The latter option is to produce a highly optimized, one time programmable accelerator for a specific application domain. However, the certain level of reconfigurability remains in the LUT array and the programmability of the Cell Array switch boxes.

Throughout its execution MC-DeF uses several metrics, thresholds, and cost functions. In Table 4, we list the name, type, and MC-DeF phase each of them used. The parameters used can be divided into two categories: those used to create a more compact and resource-efficient array and those used to create a fast and high bandwidth communication framework.

The threshold values applied can be used for design space exploration in order for the user to find a hardware solution tailored to either area or energy restrictions. This feature is also aided by the fast execution and simulation times of MC-DeF averaging below 2 minutes.

5 EXPERIMENTAL EVALUATION

In this section, we present evaluations of our framework and its individual components. For evaluation purposes, we use nine scientific applications: Hayashi Yoshida coefficient estimator [18], Mutual Information of two random variables, Transfer entropy between two processes [20], Fast Fourier transform 1D, Linear Regression, Fuzzy Logic Generator, Breast Mammogram, Locality Sensitive Hashing (LSH), and a client of a high-speed packet capture application. All the

Table 5. MC-DeF Experimental Results

	Hayashi Yoshida	**Mutual Information**	**Transfer Entropy**	**LSH**	**Capture Client**
Resources (LUT, BRAM, DSP)	(3,912,0,4)	(17,677,2,4)	(17,533,2,4)	(36,488,144,112)	(170,0,0)
DFG Nodes	270	225	199	294	118
Cell Structure	NodeEq→NodeAnd NodeAdd/Sub	NodeAdd/Sub NodeMul→NodeDiv	NodeAdd/Sub NodeMul→NodeDiv	NodeAdd/Sub NodeMul	NodeBits→NodeEq
CGRA dimensions	6×6	4×4	4×4	9×9	3x3
Clock frequency	290 MHz	270 MHz	270 MHz	270 MHz	320MHz
LUT array size	357	320	399	3,525	85
Total chip area mm^2	144	125	125	227	121
Energy consumption	22.165 J	22.245 J	22.245 J	23.519 J	32.848 J
Avg. distance/cell	3	5.4	5.6	8.8	3
Avg. FIFO size/cell	4	9.4	10.9	18.9	0
Avg. outputs/cell	1	2.5	2.5	1.2	1
Internal bandwidth	29 GB/s	43.2 GB/s	43.2 GB/s	70.2 GB/s	15.3 GB/s

	Linear Regression	**Fuzzy Logic Generator**	**Breast Mammogram**	**FFT 1D**
Resources (LUT, BRAM, DSP)	(6,841,22,10)	(26,442,66,29)	(3,410,24,0)	(68,003,285,44)
DFG nodes	87	199	109	3,209
Cell structure	NodeAdd/Sub NodeEq→NodeMux	NodeBits→ NodeAdd/Sub	NodeAnd→NodeMux NodeDiv NodeGte→NodeAnd	NodeAdd/Sub→NodeCat NodeAdd/Sub
CGRA dmensions	4×4	7×7	4×4	12×12
Clock frequency	270 MHz	250 MHz	270 MHz	270 MHz
LUT array size	881	1,073	803	15,374
Total chip area mm^2	122	184	122	293
Energy consumption	21.793 J	21.650 J	21.830 J	32.522 J
Avg. distance/cell	7.1	3.4	7	10.3
Avg. FIFO size/cell	15.3	6.6	2.5	0
Avg. outputs/cell	1.5	1	2.5	1.5
Internal bandwidth	19.4 GB/s	38 GB/s	19.4 GB/s	80.2 GB/s

application DFG files were provided by the Maxeler AppGallery site [42]. The application characteristics, i.e., resource utilization and DFG size, are presented in Table 5. In this section, we first present the results related to the final Mixed-CGRA architecture for the target applications. Individual results are also presented in this section regarding

— the Mapping Improvement Algorithm,
— the ability of our CGRA designs to map different applications in an already configured array, and
— the differences between the LUT array and the *L-i-C* design options.

Finally, we compare our MC-DeF framework and Mixed-CGRA architecture with related work in the field.

5.1 Baseline Results

To evaluate MC-DeF, we first verified its functionality using nine scientific applications' DFGs provided by Maxeler Ltd. Initially, MC-DeF determines the structure and functionality of a CGRA cell, as well as the in-cell LUT structures or the adjacent LUT array used for mapping non-cell nodes. Then, the framework continues to map the DFG nodes to cells and eventually specify the connectivity network of the design and finalize the mapping of the cells using the novel Mapping Improvement Technique. Finally, MC-DeF provides the user with area occupancy and energy consumption estimations and presents a final report to the user.

Using the finalized MC-DeF reports from each of the nine applications, we report the customized CGRA cell functionality, the size of the cell array, the mapping of non-cell nodes, the utilization of a cell in the array, i.e., average output/cell, and the achieved clock frequency. Also, MC-DeF provides the user with insight regarding the communication network's utilization; we report the average distance/cell, synchronization FIFO size/cell, and finally the internal bandwidth recorded. Energy consumption of the resulting designs is calculated using the amount of input data of each application supplied by the user. Energy consumption estimations are based on the amount of floating-point operations performed and the cell array's network energy consumption, i.e., the energy required for transferring the data through the cell array. Each cell of the cell array performs pipelined and parallel operations. MC-DeF design can be implemented either as an overlay or a stand-alone design following the architecture shown in Figure 1; for this case, the target board for our designs is a Stratix V FPGA. MC-DeF design results for all nine applications are presented in Table 5.

For each of the cell structures implemented, we calculate the energy consumption and area occupancy based on the tables presented in Section 4.4. For nodes that do not appear in this section, we use energy and area estimation as derived from simple VHDL implementation of the node, such as equality nodes. The right arrow annotation between certain nodes denotes a sub-graph inclusion in the cell, e.g., NodeEq→NodeAnd in the Hayashi Yoshida application.

For the majority of applications, the communication infrastructure is configured using 32-bit channels. MC-DeF decides on the communication infrastructure after enumerating input and output bit-widths for each node implemented in the cell array. Also, this choice is made considering the majority of operations performed for each application, which in most cases is 32-bit double precision floating point.

Results obtained verify the correct functionality of our framework. For each application, we can see that cells perform different operations, which results in different clock frequencies achieved. The highest frequency, 320 MHz, is achieved in the smallest application, High-Speed Capture Client, while Fuzzy Logic Generator records the lowest frequency, 250 MHz. Also, fluctuations in area occupancy and energy consumption estimations are based on the type of operations performed and the size of the cell array; the most expensive application in terms of energy and area is the FFT one. Some interesting observations we can make by analyzing the results shown in the tables is the energy and area tradeoff evident in the Hayashi Yoshida application. As seen in Table 5. Hayashi Yoshida has similar energy consumption with the Mutual Information and Transfer Entropy application. This is due to the larger cell array size of Hayashi Yoshida compared to the other two applications and the fact that the Equality and logical AND operations are 64 bits wide, thus utilizing two 32-bit circuits for each operation.

5.2 L-i-C vs LUT Array Impact

The two approaches regarding mapping non-cell nodes have vastly different architectures regarding both mapping the nodes and routing. First, it is necessary to distinguish the LUTs used to map rogue nodes (on the LUT array) and the LUTs used for memory purposes (on Memory LUTs). It is established that for buffers or even small memory FIFOs the mapping tools will not always use BRAMs but LUT structures. These memory structures are placed, by MC-DeF, inside the cells using them in order to minimize the communication overhead induced.

Table 6. LUTs-in-Cell vs. LUT-Array Configurations

	Hayashi Yoshida	**Mutual Information**	**Transfer Entropy**
LUT array	361	429	473
LUT percentage	9.22	2.42	2.69
Memory LUTs	562	4,481	2,046
LUT + MemLUT percentage	23.59	27.77	14.36
Avg. L-i-C array	19	27	30
Max. L-i-C array	26	104	104
Avg. distance/cell	3.7	6.7	7
Internal bandwidth	37.5 GB/s	48.3 GB/s	48.3 GB/s

The migration of the LUT array inside the cells refers only to LUTs used for nodes and not memory LUTs. In Table 6, we present the results comparing the two different LUT configurations. The cell LUT arrays are currently of varying size, so we report an average L-i-C array size and a maximum one. A part of the future work is to have a unified size for all the cell LUT arrays and create larger ones when necessary by merging arrays from different cells, similar to Stitch [41]. The varying size of *L-i-C* arrays is the reason why designs with the *L-i-C* option are not able to accommodate various applications.

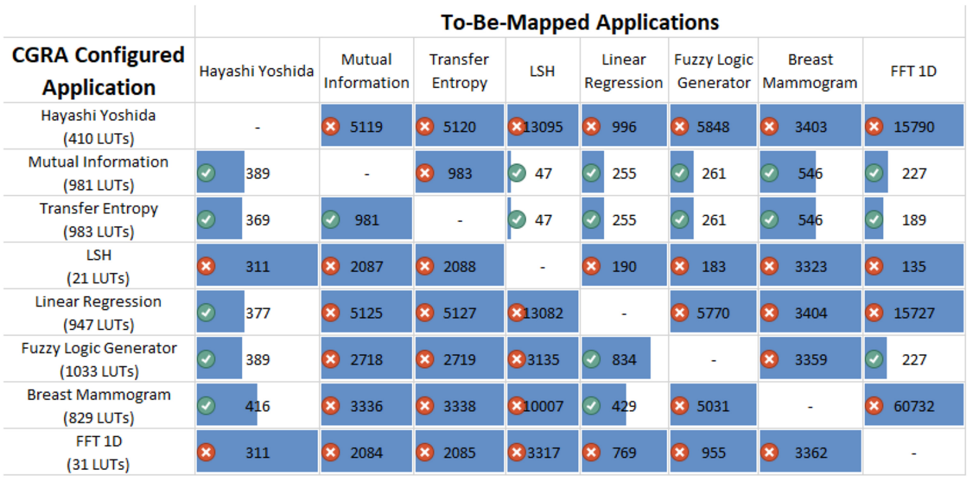
The overhead induced by the individual LUT-array network is observed in the distance per cell metric. However, it does not correlate directly to a decrease in the design's internal bandwidth, because the operations performed by a single cell are increased. This is the case with the Hayashi-Yoshida application, where despite the fact that the distance/cell metric is increased, the overall bandwidth is also increased due to more operations performed simultaneously and transmitted in parallel with the cell network via the dedicated LUT network.

Finally, we should note that through the *CSFD* phase of MC-DeF, the created cells are highly customized and tailored to the application's needs. The percentage of LUTs used for the implementation of an application, through the use of *CSFD*-related techniques and algorithms, is not over 10%. Even when including LUTs for memory structures, the largest percentage is for Mutual Information—27.77%.

5.3 Flexible Mapping of New Applications on Existing CGRAs

It is often the case that between application runs, the user is not able to re-run the whole framework and create an application-customized CGRA design. MC-DeF offers the ability to the user to map a different application in an already configured CGRA. This operation is carried out in a time-multiplexed fashion by swapping applications that are not originally configured for the existing CGRA architecture. The new application uses available logic in the cells that matches its operational demands, and utilizes the LUT array as needed.

In the MC-DeF case, we conduct experiments and record measurements of the mapping ability, for each application generated CGRA, to map all the other applications. The NxN table presented in Figure 5 denotes whether or not an application can be mapped on the configured CGRA. In the y-axis we see the applications used for the CGRA configuration and on the x-axis the to-be-mapped applications. In order for an application to be mapped, the LUT array provided originally must be large enough to accommodate the logic not able to be mapped on the cell array, denoted by the ✗ and ✓ symbols. Also, we present the needed LUTs for each application in order to be mapped on the CGRA and based on how much each square is filled, we can see how



CGRA Configured Application	To-Be-Mapped Applications: Hayashi Yoshida	Mutual Information	Transfer Entropy	LSH	Linear Regression	Fuzzy Logic Generator	Breast Mammogram	FFT 1D
Hayashi Yoshida (410 LUTs)	-	✗ 5119	✗ 5120	✗ 13095	✗ 996	✗ 5848	✗ 3403	✗ 15790
Mutual Information (981 LUTs)	✓ 389	-	✗ 983	✓ 47	✓ 255	✓ 261	✓ 546	✓ 227
Transfer Entropy (983 LUTs)	✓ 369	✓ 981	-	✓ 47	✓ 255	✓ 261	✓ 546	✓ 189
LSH (21 LUTs)	✗ 311	✗ 2087	✗ 2088	-	✗ 190	✗ 183	✗ 3323	✗ 135
Linear Regression (947 LUTs)	✓ 377	✗ 5125	✗ 5127	✗ 13082	-	✗ 5770	✗ 3404	✗ 15727
Fuzzy Logic Generator (1033 LUTs)	✓ 389	✗ 2718	✗ 2719	✗ 3135	✓ 834	-	✗ 3359	✓ 227
Breast Mammogram (829 LUTs)	✓ 416	✗ 3336	✗ 3338	✗ 10007	✓ 429	✗ 5031	-	✗ 60732
FFT 1D (31 LUTs)	✗ 311	✗ 2084	✗ 2085	✗ 3317	✗ 769	✗ 955	✗ 3362	-

Fig. 5. Table that shows the needed LUTs and the ability to map each application on each already-configured CGRA. The coverage of the blue bar shows how many of the available LUTs in the adjacent LUT array are used when another application is mapped on the configured CGRA. An ✗ denotes the application cannot be mapped.

Table 7. Percentage of Mapped Applications

Available LUTs	Applications: Hayashi Yoshida	Mutual Information	Transfer Entropy	LSH	Linear Regression	Fuzzy Logic Generator	Breast Mamm.	FFT 1D
0	0	86	100	0	15	42	28	0
1,500	15	100	100	42	15	42	28	42
3,500	42	100	100	100	28	100	57	100
Total LUTs needed for 100% mapping	15,790	983	983	2,088	15,727	3,440	60,792	3,362

under-utilized the LUT array is. A fully colored cell in the table means that the new application needs more LUTs compared to the ones already present, e.g., the Mutual Information application needs 5,119 LUTs to be mapped on a Hayashi Yoshida configured CGRA, while the available LUTs are only 410, thus the corresponding square is full and the ✗ symbol is showing in the square. By loading the configured CGRA architecture's and the new application's .xml files in MC-DeF, the framework is able to create a mapped and routed design of the new application. This process of swapping applications in an existing CGRA cannot be used while simultaneously using the L-i-C design option. Additionally, the user has to plan ahead for this situation by giving more space to the adjacent LUT array. The added overhead in terms of unused LUT resources is proportional to the generality added in the design. In Table 7, we see the *generality percentage*, i.e., the percentage of the total applications able to be placed on a configured CGRA for all reference applications, e.g., a Mutal Information configured CGRA is able to map 89% of the other applications without additional LUTs in the configured LUT array. Additionally, we present the generality percentages when there are more than the original deigned LUTs available (1,500 and 3,500) and, finally, the minimum number of LUTs needed in order for an application-configured CGRA design to be able to map all the other target applications.

A similar attempt in making one CGRA design and adapting different applications on it is done in [40]. There the authors try to create an application domain-specific CGRA and through a Generality metric determine the percentage of applications in a certain domain that can be mapped on it. In our studies, so far we have not employed an application domain approach in the creation of the CGRA specification. However, in Table 7 we can see the ability of MC-DeF to build scalable architectures that, provided enough resources, are able to accommodate applications from different domains. A key observation to be made is the importance of the composition of a CGRA cell; during the *CSFD* phase, MC-DeF tags for inclusion not only node sequences, as is the case in [40], but also individual nodes. This provides the cell with more versatility and the reason why, provided enough resources, the generality percentage of MC-DeF is able to reach 100.

5.4 Mapping Improvement

The next set of figures presents the correct functionality of the **Mapping Improvement** technique. As stated in Section 4.3, the Mapping Improvement algorithm implemented is used to minimize the average distance/cell metric instead of the average synchronization FIFO size one. To make this decision, we conducted experiments where both metrics were the minimization target and recorded the variance of the other.

In the case where the FIFO size metric was the minimization target, the average distance/cell metric showed a variance of 7.8 for Mutual Information, 5.7 for Transfer Entropy, and 4.4 for Hayashi Yoshida. In the opposite case, the variance of the average FIFO size metric was 1.5, 1.1, and 1.01, respectively. Since the variance of the average distance/cell metric is larger, the possible improvement by minimizing it is also larger. This led us to choose the average distance/cell metric as the minimize target. The Improved Mapping Technique terminates its execution after performing 100 cell movements. In the experiments performed, we observed that beyond that point there was very little, if any, improvement in the average distance/cell metric. This number is not fixed in the framework's code but is able to change according to the user and the application's size, with an added increase in the framework's execution time. In Figures 6–8, we record the values of the *average distance/cell* and *average synchronization FIFO size* metrics over the course of the cell movements performed for our target applications. The orange line shows the trend of the distance/cell metric during its descent, while gray dots are values that are larger than the current minimum. Finally, the blue line is the progress of the sync. FIFO size metric. The x-axis is the number of cell movements performed ending at 100 and the y-axis is the average value among all the cells of the two metrics recorded. The number of cell movements performed is chosen by the user and depends on the design time effort spent while MC-DeF searches for an optimal mapping solution.

We observe that the point where a minimum is found is arbitrary and differs between the three applications. For example, the Mutual Information hits its minimum distance/cell value after the 45th and the 100th cell movement, while Transfer Entropy records its value after the 99th. Additionally, compared to the initial mapping for each application, we can see a 64.2%, a 70%, and a 51.2% decrease in the Avg. Distance/Cell for the Transfer Entropy, Hayashi Yoshida, and Mutual Information applications, respectively.

5.5 Comparisons

In Section 2, we referenced several works that also propose CGRA designs. Performing fair and direct comparisons between different CGRA architectures is a complex procedure; a main obstacle we encountered was the fact that each CGRA architecture had been implemented and/or targeted on a different FPGA fabric or utilized a different overlay architecture.

Moreover, the applications used to evaluate MC-DeF do not appear in any of the related works examined for comparison purposes. To achieve a fair and meaningful comparison, we followed the

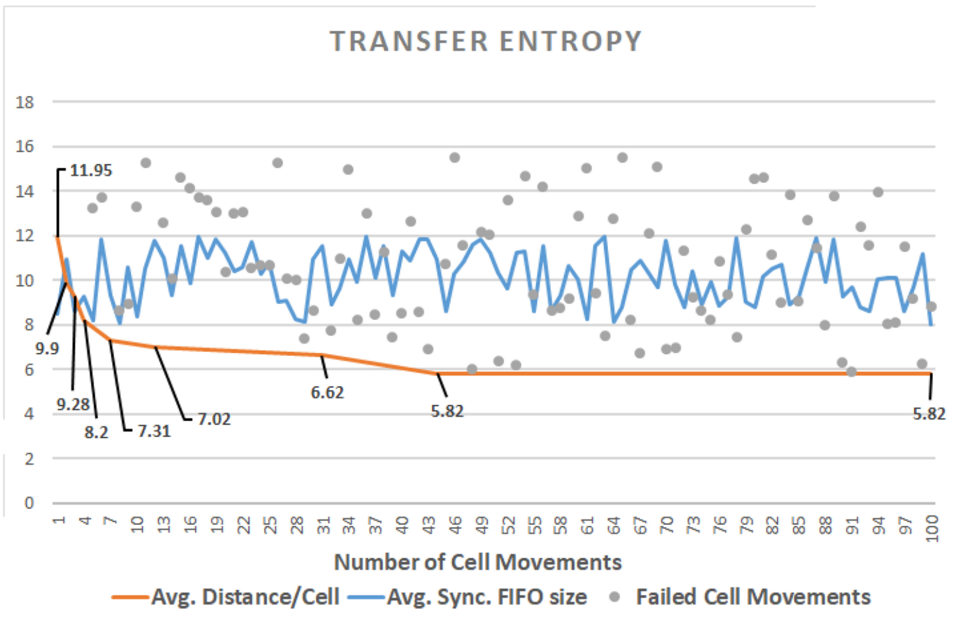


Fig. 6. Mapping Improvement Algorithm for Transfer Entropy.

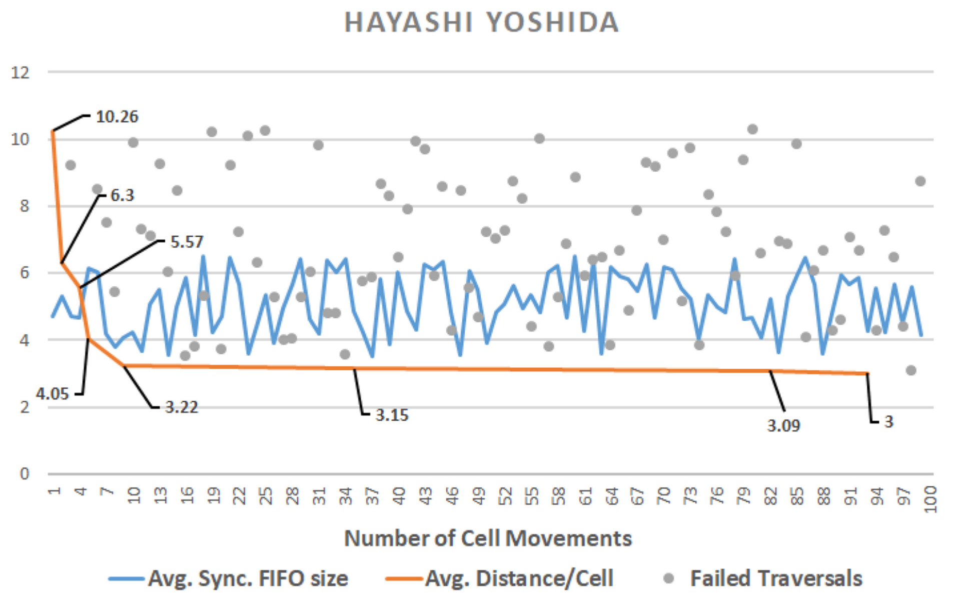


Fig. 7. Mapping Improvement Algorithm for Hayashi Yoshida.

methodology presented in [25]. We compare a baseline scenario architecture created by MC-DeF, i.e., the CGRA implemented for the Hayashi Yoshida application, with *Intermediate Fabrics, Intermediate Fabrics (opt), DySER, and DECO*; for all the compared architectures the numbers presented are according to their best performing application. Our comparisons are based on generic metrics:

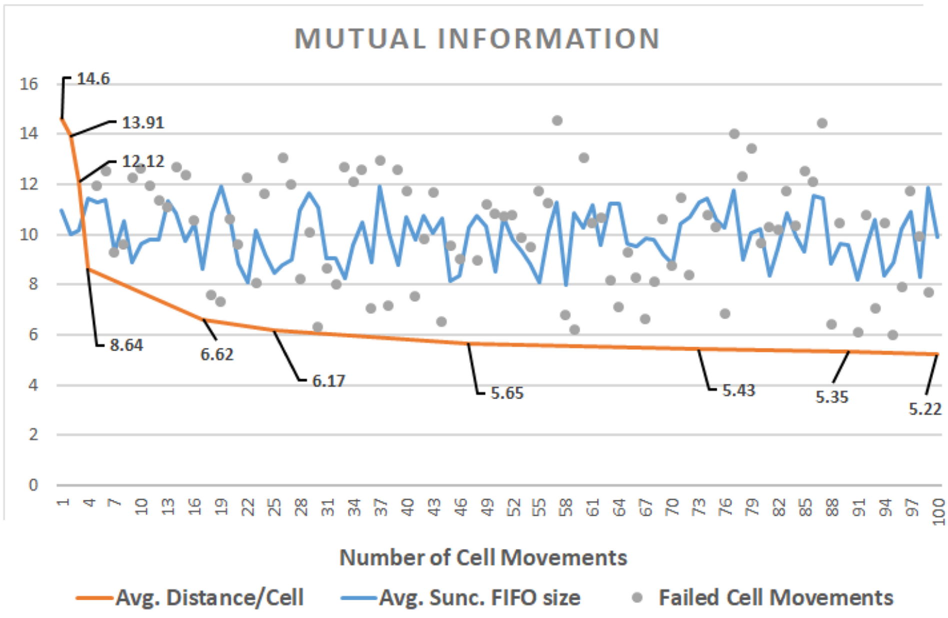


Fig. 8. Mapping Improvement Algorithm for Mutual Information.

Table 8. Quantitative Comparison of CGRAs

Resource	IF	IF (opt.)	DySER	DECO (DSP)	MC-DeF	MC-DeF+DSP
CGRA grid	14×14	14×14	6×6	20 (cone)	6×6	6×6
Frequency	131	148	175	395	290	290
Total OPs	196	196	36	60	108	108
Peak GOPs	25.6	29	6.3	23.7	31.3	31.3
LUT used	91 K	50 K	48 K	10 K	15 K	13 K
DSPs used	196	196	36	20	0	36
LUTs/GOPs	3,550	1,725	7,620	430	482	415

clock frequency achieved, total operations carried out in parallel on the array, peak giga operations per second on a fully utilized array, and resource utilization.

In [25], the authors encountered the same obstacle when trying to perform fair and direct comparisons between different CGRA architectures. The solution provided was a new comparison metric, LUTs/GOPs. This metric represents the interconnect resource used per unit peak throughput, giving the ability to quantify the area overhead of the overall interconnect architectures irrespective of the different FU/cell implementation. The results obtained by comparing our framework with state-of-the-art related works are presented in Table 8.

The FPGA devices for each CGRA architecture are as follows: *Intermediate Fabrics* architecture is implemented in an Altera Stratix III E260 FPGA; *DySER* and *DECO* are both implemented in a Xilinx Zynq XC7Z020 device. A uniqueness observed in *DECO* is that the cells are arranged in a cone shape so X, Y array size is not applicable. As a reference design for our comparisons, we consider a CGRA size 6×6 with three operations/cell operating at a maximum *frequency* of 290 MHz, the actual frequency achieved for the Hayashi-Yoshida application (Table 5). To calculate the number of operations performed by our architecture, we multiply the clock frequency achieved with the total operations performed by each cell when the CGRA is fully utilized. In all of the cases the cell is considered to perform all available operations per clock cycle.

An advantage of MC-DeF compared to related work is its ability to map multiple operational nodes from the application's DFG in a single cell. Since operations in the cell are parallel and pipelined, the total operations performed in the cell is equal to the number of original DFG nodes in it, e.g., the Transfer Entropy design performs three operations/cell at each cycle. As the contents of the cell, as defined by our framework, matches the needs and structure of the application DFG nodes, both the utilization and the processing throughput of the corresponding circuits within the cell are high. This is an advantage compared to related work where the basis of a cell is a generic large FU/ALU performing at maximum one or two operations per cycle. This also means that our designs are more compact, thus reducing resource utilization and energy consumption.

Currently, MC-DeF creates a CGRA architecture based on a generic VHDL library of node implementations for the cell's structure and functionality. This implementation choice under-utilizes the DSP circuits offered by the FPGA device. However, by forcing MC-DeF to include one DSP per cell of fixed hardware logic and reconfigurable LUTs as is our intention. This also reduces the LUTs used by the whole architecture. The above implementation choice is dubbed MC-DeF (+DSPs) in Table 8 and records the best results in LUTs/GOPs metric over all compared related work.

In terms of peak performance (GOPs), the standard *IF* architectures is 1.22× worse than MC-DeF+DSP, while being 7× larger in terms of LUTs used. *IF (opt.)* design again is outperformed by MC-DeF+DSP in terms of GOPs by a factor of 1.07×, but still is quite resource consuming, utilizing 4× more LUTs than the MC-DeF+DSP design. Compared to *DySER*, an MC-DeF design is able to perform more operations and also have a better LUTs/GOPs metric. In fact, an unoptimized MC-DeF design has the second lowest LUTs/GOPs metric approaching an almost ideal interconnect area overhead.

From the related works used for comparison purposes, we found that *DECO* is the best performing one. However, the superiority of DECO is mainly due to the high clock frequency achieved, a result of the fact that DECO is a DSP-only CGRA architecture. There are two ways for MC-DeF to achieve a better LUTs/GOPs metric. The first one is to increase the number of peak GOPs by including more operations in a single cell. This is feasible due to the customization performed during the *CSFD* phase of our framework. We also note that the applications we used to evaluate MC-DeF are larger (in terms of nodes) than the ones used in other works. However, the baseline version of MC-DeF achieves comparable results in the LUTs/GOPs metric.

Another option is to force the mapping tools of MC-DeF to utilize at least one DSP circuit per cell. For example, for the implementation of an adder, MC-DeF would not opt for a hybrid LUT+DSP implementation but would use only DSPs to achieve even higher clock frequencies while reducing the amount of LUT resources used. To improve the LUTs/GOPs metric, we implemented an improved version of MC-DeF using DSP blocks as computational units in our cell array. This reduced the LUTs utilization, thus increasing the LUTs/GOPs metric. However, the presence of slower LUT-based nodes in the cell prevented MC-DeF from achieving really high clock frequencies. For the optimized MC-DeF+DSP design, we used 13K LUTs and 36 DSP blocks, which dropped the LUTs/GOPs metric to 415, the best among related works we compared against.

By reviewing the results presented in Table 8, we were able to identify the advantages of MC-DeF over related work:

— MC-DeF is able to create highly customized CGRA cells that match the application computational needs.

- Universality of operations performed over an application domain allow the user to map and execute other applications on an existing array (Section 5.3).
- Superiority of MC-DeF in terms of GOPs performed even when compared with larger designs.
- Better LUT resource utilization when compared with same array-sized designs like DySER.
- The best recorded LUTs/GOPs metric when DSPs are used explicitly on the cell's implementation.

6 CONCLUSIONS

In this article. we presented a complete CGRA definition framework for dataflow applications. MC-DeF performs all the required steps to create a complete CGRA design. Using an application's dataflow graph, MC-DeF performs (a) cell structure and functionality definition, (b) DFG node mapping in the resulting cell array, (c) routing of the array and the reconfigurable logic, and (d) area occupancy and energy consumption estimations assuming a 7 nm lithography technology.

MC-DeF offers flexibility support through the use of LUT structures, either in an adjacent LUT array or included in the cell structure, a unique feature among related works in the field. This level of reconfigurability introduced in the CGRA architecture allows for efficient mapping of irregular computations that do not fit the cell computation features. MC-DeF uses a variety of cost functions, threshold values, and metrics that can be adjusted by the designer to create designs that reach the desired cost and performance requirements and/or area and energy restrictions.

Also, we explained in detail each phase of our framework and introduced three new features: first, the ability of MC-DeF to flexibly and efficiently map similarly structured applications on an existing CGRA, an extension to include small LUT structure in the CGRA cells thus increasing the maximum operations performed in a cell, and finally, a novel Mapping Improvement Algorithm able to decrease the average distance per cell up to 70% when compared to the original placement minimizing communication overhead.

We evaluated MC-DeF using nine scientific applications and provided CGRA designs for all of them. The highest frequency recorded in these experiments was 320 MHz for a high-speed capture client implementation, while the largest internal bandwidth achieved was 80 GB/s for the FFT application. Comparisons to the state of the art shows that MC-DeF performs favorably in terms of giga operations performed per second even when compared with much larger designs in terms of CGRA size. MC-DeF uses less resources than most of the compared architectures, and better utilizes the underlying architecture recording the second best LUT/GOPs rating. During the comparison experiments, we created a resource-optimized version of our framework, MC-DeF+DSP, that creates designs forced to use one DSP per cell to decrease LUT utilization. The new design created recorded the best LUTs/GOPs metric (415) compared to related works.

A key issue that we plan on working on in the future is the communication infrastructure. Drawing inspiration from DECO's coned-shape CGRA and considering the uniqueness of each application's DFG shape and structure, we plan to explore and evaluate alternative network topologies, connectivity, and so forth. Additionally, we want to observe how our designs scale in terms of resources and performance, either by creating duplicate CGRAs that execute in parallel or by creating a larger CGRA able to utilize the entirety of the available hardware for better bandwidth and execution times. Finally, we want to create a suite of available Mixed-CGRA designs from various benchmark scientific applications.

REFERENCES

[1] E. Ahmed and J. Rose. 2004. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 3 (March 2004), 288–298.

[2] Mythri Alle, Keshavan Varadarajan, Alexander Fell, Ramesh Reddy C., Nimmy Joseph, Saptarsi Das, Prasenjit Biswas, Jugantor Chetia, Adarsh Rao, S. K. Nandy, and Ranjani Narayan. 2009. REDEFINE: Runtime reconfigurable polymorphic ASIC. *ACM Transactions on Embedded Computing Systems* 9, 2, Article 11 (Oct. 2009), 48 pages.

[3] G. Ansaloni, P. Bonzini, and L. Pozzi. 2011. EGRA: A coarse grained reconfigurable architectural template. *IEEE Transactions on Very Large Scale Integration Systems* 19, 6 (June 2011), 1062–1074.

[4] J. Chang et al. 2017. 12.1 A 7nm 256Mb SRAM in high-k metal-gate FinFET technology with write-assist circuitry for low-VMIN applications. In *2017 IEEE International Solid-State Circuits Conference (ISSCC'17)*. 206–207.

[5] George Charitopoulos and Dionisios N. Pnevmatikatos. 2018. DARSA: A dataflow analysis tool for reconfigurable platforms. In *18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'18)*. 65–72.

[6] George Charitopoulos and Dionisios N. Pnevmatikatos. 2020. A CGRA definition framework for dataflow applications. In *Applied Reconfigurable Computing*. Springer International Publishing, Cham.

[7] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson. 2017. CGRA-ME: A unified framework for CGRA modelling and exploration. In *2017 IEEE 28th International Conference on Application-Specific Systems, Architectures and Processors (ASAP'17)*. 184–189. DOI: https://doi.org/10.1109/ASAP.2017.7995277

[8] N. Clark, Hongtao Zhong, and S. Mahlke. 2003. Processor acceleration through automated instruction set customization. In *36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*. 129–140.

[9] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou. 2014. A fully pipelined and dynamically composable architecture of CGRA. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. 9–16. DOI: https://doi.org/10.1109/FCCM.2014.12

[10] J. Coole and G. Stitt. 2010. Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'10)*. 13–22.

[11] Bill Dally. 2015. Challenges for Future Computing Systems. *Presentation in HiPEAC Conference.*

[12] Bjorn De Sutter, Praveen Raghavan, and Andy Lambrechts. 2019. Coarse-grained reconfigurable array architectures. In *Handbook of Signal Processing Systems*. Springer, 427–472.

[13] Carl Ebeling, Darren C. Cronquist, and Paul Franklin. 1996. RaPiD—Reconfigurable pipelined datapath. In *The 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers (FPL'96)*. 126–135.

[14] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. GraMi: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment* 7, 7 (March 2014), 517–528.

[15] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim. 2012. DySER: Unifying functionality and parallelism specialization for energy-efficient computing. *IEEE Micro* 32, 5 (Sept. 2012), 38–51.

[16] V. Govindaraju, C. Ho, and K. Sankaralingam. 2011. Dynamically specialized datapaths for energy efficient computing. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 503–514.

[17] Reiner Hartenstein. 2001. Coarse grain reconfigurable architecture (embedded tutorial). In *The 2001 Asia and South Pacific Design Automation Conference (DAC'01)*. ACM, 564–570.

[18] Takaki Hayashi and Nakahiro Yoshida. 2005. On covariance estimation of non-synchronously observed diffusion processes. *Bernoulli* 11, 2 (April 2005), 359–379.

[19] Wen-Hsiang Hu, Seung Eun Lee, and Nader Bagherzadeh. 2008. DMesh: A diagonally-linked mesh network-on-chip architecture. *Network on Chip Architectures* (2008), 14.

[20] Konstantinos Iordanou, Sofia Maria Nikolakaki, Pavlos Malakonakis, and Apostolos Dollas. 2018. A performance evaluation of multi-FPGA architectures for computations of information transfer. In *18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'18)*. 1–9.

[21] M. Jacobsen, P. Meng, S. Sampangi, and R. Kastner. 2014. FPGA accelerated online boosting for multi-target tracking. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. 165–168.

[22] A. K. Jain, S. A. Fahmy, and D. L. Maskell. 2015. Efficient overlay architecture based on DSP blocks. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. 25–28.

[23] Abhishek Kumar Jain, Xiangwei Li, Suhaib A. Fahmy, and Douglas L. Maskell. 2016. Adapting the DySER architecture with DSP blocks as an overlay for the Xilinx Zynq. *SIGARCH Computer Architecture News* 43, 4 (April 2016), 28–33. DOI: https://doi.org/10.1145/2927964.2927970

[24] A. K. Jain, X. Li, P. Singhai, D. L. Maskell, and S. A. Fahmy. 2016. DeCO: A DSP block based FPGA accelerator overlay with low overhead interconnect. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'16)*. 1–8.

[25] A. K. Jain, D. L. Maskell, and S. A. Fahmy. 2016. Are coarse-grained overlays ready for general purpose application acceleration on FPGAs? In *2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing,*

14th International Conference on Pervasive Intelligence and Computing (DASC/PiCom/DataCom/CyberSciTech'16). 586–593.

[26] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco. 2011. GPUs and the future of parallel computing. *IEEE Micro* 31, 5 (Sept. 2011), 7–17.

[27] I. Kuon and J. Rose. 2007. Measuring the Gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2 (2007), 203–215.

[28] Aaron Landy and Greg Stitt. 2012. A low-overhead interconnect architecture for virtual reconfigurable fabrics. In *The 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'12).* Association for Computing Machinery, New York, NY, 111–120. DOI: https://doi.org/10.1145/2380403.2380427

[29] C. Liu, H. Ng, and H. K. So. 2015. QuickDough: A rapid FPGA loop accelerator design framework using soft CGRA overlay. In *2015 International Conference on Field Programmable Technology (FPT'15).* 56–63. DOI: https://doi.org/10.1109/FPT.2015.7393130

[30] C. Liu, C. L. Yu, and H. K. So. 2013. A soft coarse-grained reconfigurable array based high-level synthesis methodology: Promoting design productivity and exploring extreme FPGA frequency. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines.* 228–228. DOI: https://doi.org/10.1109/FCCM.2013.21

[31] D. Liu et al. 2018. Data-flow graph mapping optimization for CGRA with deep reinforcement learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018), 1–1.

[32] Sen Ma, Zeyad Aklah, and David Andrews. 2016. Just in time assembly of accelerators. In *The 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16).* Association for Computing Machinery, New York, NY, 173–178. DOI: https://doi.org/10.1145/2847263.2847341

[33] K. T. Madhu, S. Das, S. Nalesh, S. K. Nandy, and R. Narayan. 2015. Compiling HPC kernels for the REDEFINE CGRA. In *IEEE 17th International Conference on High Performance Computing and Communications, and 12th International Conference on Embedded Software and Systems.* 405–410.

[34] A. Niedermeier, Jan Kuper, and Gerard J. M. Smit. 2014. A dataflow inspired programming paradigm for coarse-grained reconfigurable arrays. In *Reconfigurable Computing: Architectures, Tools, and Applications.* Springer International Publishing, Cham, 275–282.

[35] O. Pell and V. Averbukh. 2012. Maximum performance computing with dataflow engines. *Computing in Science Engineering* 14, 4 (July 2012), 98–103.

[36] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini. 2015. PULP: A parallel ultra low power platform for next generation IoT applications. In *2015 IEEE Hot Chips 27 Symposium (HCS'15).* 1–39. DOI: https://doi.org/10.1109/HOTCHIPS.2015.7477325

[37] Mainak Sen et al. 2007. Dataflow-based mapping of computer vision algorithms onto FPGAs. *EURASIP Journal on Embedded Systems* 2007, 1 (Jan. 2007), 049236.

[38] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz. 2013. Reconfigurable computing in next-generation automotive networks. *IEEE Embedded Systems Letters* 5, 1 (2013), 12–15.

[39] T. Standaert et al. 2016. BEOL process integration for the 7 nm technology node. In *2016 IEEE International Interconnect Technology Conference/Advanced Metallization Conference (IITC/AMC'16).* 2–4.

[40] M. Stojilovi ć, D. Novo, L. Saranovac, P. Brisk, and P. Ienne. 2013. Selective flexibility: Creating domain-specific reconfigurable arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 5 (May 2013), 681–694.

[41] C. Tan, M. Karunaratne, T. Mitra, and L. Peh. 2018. Stitch: Fusible heterogeneous accelerators enmeshed with many-core architecture for wearables. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18).* 575–587. DOI: https://doi.org/10.1109/ISCA.2018.00054

[42] Nemanja Trifunovic, Veljko Milutinovic, Nenad Korolija, and Georgi Gaydadjiev. 2016. An AppGallery for dataflow computing. *Journal of Big Data* 3, 1 (2016), 4.

[43] B. S. C. Varma, K. Paul, and M. Balakrishnan. 2013. Accelerating 3D-FFT using hard embedded blocks in FPGAs. In *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems.* 92–97.

[44] Xilinx 2018. *7 Series FPGAs Data Sheet: Overview.* Xilinx. Rev. 2.6.

[45] S. Yin, D. Liu, L. Sun, L. Liu, and S. Wei. 2017. DFGNet: Mapping dataflow graph onto CGRA by a deep learning approach. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS'17).* 1–4.

Received July 2020; revised December 2020; accepted January 2021