ICED: An Integrated CGRA Framework Enabling DVFS-Aware Acceleration

Cheng Tan[†§], Miaomiao Jiang[¶§], Deepak Patil[§], Yanghui Ou[‖], Zhaoying Li[‡], Lei Ju[¶], Tulika Mitra[‡], Hyunchul Park[†], Antonino Tumeo[*], Jeff Zhang[§]

[†]Google, [§]Arizona State University, [¶]Shandong University, [‖]Cornell University,
[‡] National University of Singapore, [*]Pacific Northwest National Laboratory
[†]{chengtan, parkhc}@google.com, [§]{chengtan, mjiang58, dppatil3, jeffzhang}@asu.edu,
[¶]{miaomiaojiang, julei}@sdu.edu.cn, [‖]yo96@cornell.edu, [‡]{zhaoying, tulika}@comp.nus.edu.sg,
[*]antonino.tumeo@pnnl.gov

***Abstract*—Coarse-grained reconfigurable arrays (CGRAs) are a promising solution to enable energy-efficient acceleration of applications from different domains. By leveraging reconfiguration at the functional level, they can adapt to significantly different computational patterns. However, the relationships of voltage and frequency with the utilization of CGRA resources and the dynamic management of them are not well explored, leading to inefficient designs.**

CGRAs have also been successful in accelerating data-dependent streaming applications. However, in these applications, the execution time of each kernel in the pipeline might dynamically vary depending on the characteristics of the input. This also leads to under-utilization of resources for the dynamically changing kernels that do not limit the application throughput. DVFS can also improve energy efficiency for these applications by dynamically changing the voltage and frequency levels of tiles that host non-performance-constraining kernels.

This paper proposes ICED – an integrated DVFS-aware framework to map applications on CGRAs that support power islands. ICED proposes a CGRA architecture supporting DVFS islands at varying granularity (from a single tile to a group of tiles) and the related DVFS-aware compilation and mapping toolchain. ICED is the first work that introduces DVFS support for spatio-temporal CGRAs at power-island levels. The experimental evaluation shows that ICED improves average utilization by 2.3× and energy-efficiency by 1.32× over a conventional CGRA. With streaming applications, ICED can achieve up to 1.26× energy-efficiency compared with a state-of-the-art CGRA that introduces partial dynamic reconfiguration to adapt to variations in kernels' throughput.

I. Introduction

Coarse-grained reconfigurable arrays (CGRAs) are accelerators composed of tiles, containing one or more functional units (FU), interconnected through an on-chip network. They usually are loosely coupled to a general-purpose central processing unit and enable the acceleration of a variety of computational patterns through their reconfigurability [21]. They have been shown to provide energy-efficient acceleration of applications from different domains, including machine learning (ML) [17], [38], high-performance computing (HPC) [9], and embedded systems [15].

With all reconfigurable devices and, in particular, with CGRAs, the compiler infrastructures play a critical role in achieving the theoretical peak performance provided by the hardware resources. A widely used approach maps the data-flow graph (DFG) of a kernel (typically a performance critical loop nest of the application) on the time-extended Modulo Routing Resource Graph (MRRG) [15], [16], [22] of the spatial-temporal CGRA, trying to minimize the initiation interval (II), i.e., the delay in clock cycles needed to launch a new loop iteration. However, mismatches between the size of the kernels and available computing and communication resources could lead to under-utilization of the reconfigurable substrate. For example, in the simple case of a small kernel with only a few DFG nodes (e.g, tens) and a large CGRA with many homogeneous tiles (e.g., hundreds), only the tiles allocated to the kernel are utilized, while the others remain idle [35]. In a more complex case, loop-carried dependencies can complicate the mapping and lead to imbalanced tile utilization. In fact, even if all tiles are mapped to a DFG node, the resources can still be under-utilized due to a high II. This can happen if the DFG has a long recurrence cycle due to loop-carried dependencies, thus requiring a large MRRG for the mapping, which can be viewed as replicating resources of the CGRA for II times. In both cases, the end result is that energy is wasted on tiles with low or no utilization. One possible solution to improve energy efficiency and utilization is to apply dynamic voltage and frequency scaling (DVFS) to each tile, or subsets of tiles (islands).

Previous works [19], [25], [29] also show that data-dependent streaming applications composed of multiple kernels can lead to under-utilization of resources. In fact, in these applications, the execution time of a kernel can vary with the input data set, leading to an imbalanced pipeline of kernels where the bottleneck can dynamically change. If the resource allocation for each kernel is fixed, the kernels that currently are not a bottleneck (meaning that they could execute faster if not limited by other ones) will under-utilize their resources and have tiles with long idle times.

In this paper, we take the first steps towards solving all these challenges by introducing ICED — an Integrated CGRA framework Enabling DVFS-aware acceleration. The main con-

tributions of this work are:

- a DVFS-aware CGRA design to enable the configuration of voltage and frequency in CGRA tiles. To the best of our knowledge, ICED is the first work that introduces DVFS into a spatio-temporal CGRA by supporting islands of arbitrary size;
- a corresponding DVFS-aware compilation toolchain to map application kernels and full applications on the ICED CGRA architecture;
- the impact of the ICED approach on data streaming applications, where dynamic frequency and voltage scaling can be applied to (changing) kernels that are not throughput-limiting;
- the experimental evaluation of ICED with kernels from very different domains (embedded, machine learning, and high-performance computing) and full streaming applications.

II. Background and Motivation

This section provides background on how CGRAs are typically used to accelerate applications and illustrates the motivations behind ICED.

A. Background

Figure 1 shows how a synthetic kernel is mapped onto a 4×4 CGRA. We simplify the DFG by flattening the nested-loop and removing the address calculation node before the `ld` operation. Control flow is converted into data-flow using partial predication [12]. The loop has an II of 4, meaning that it takes 4 cycles to start a new loop iteration. This provides a speedup of 2.75× (i.e., #nodes ÷ II = 11 ÷ 4) in terms of execution cycles over a single-issue in-order CPU. However, the utilization of the tiles is not balanced. For example, `tile9` (highlighted with a red circle) is only active at `cycle1` and `tile15` is always idle. This provides opportunities to lower voltage and/or frequency of `tile9` to improve utilization and reduce energy consumption. Note that the execution of the data-flow is predicate-based, for example, the first `n8` is executed at `cycle1` but its output is invalid (as there are no `n5` and n6 executed yet so its inputs are invalid). The first valid execution of `n8` is actually at `cycle5`.

Imbalanced Utilization across Tiles – Figure 2 shows the average utilization of the tiles of CGRAs in various sizes, across a set of representative kernels from different domains (see Table I). Note that the utilization of a tile considers both the FUs (for computation) and the crossbar (for communication) as detailed in the architecture section. We can see that the average utilization across tiles usually decreases when running on larger CGRAs. However, this is not always the case when the DFG size increases (i.e., by unrolling). For example, the `spmv` kernel is composed of more DFG nodes than `conv` and `relu` (as shown in Table I), but it results in a lower utilization than them (the blue bar) when running on a 6×6 CGRA. This happens because the II of `spmv` increases after unrolling due to its loop-carried dependency. The same thing happens to `gemm`. Therefore, there is an opportunity to improve the overall energy efficiency by slowing down some of the tiles (i.e., by decreasing their voltage and frequency).

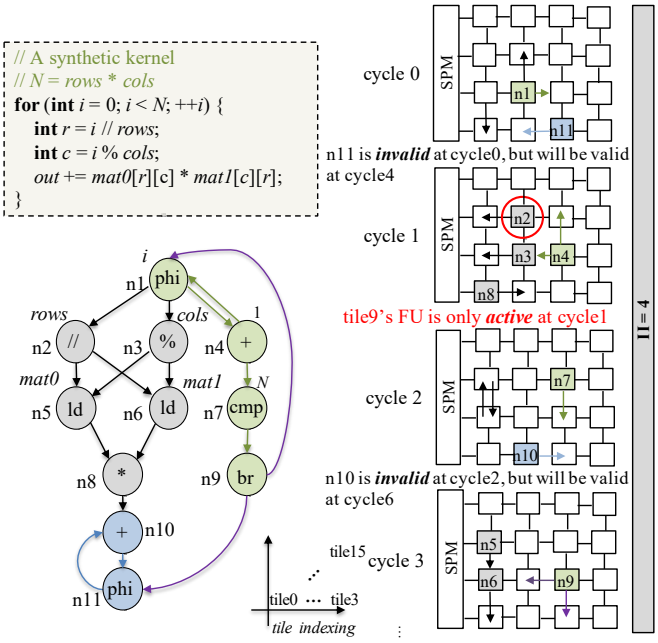


Fig. 1: A synthetic kernel, DFG, and the conventional mapping on a 4×4 CGRA – The DFG nodes `n1`, `n4`, `n7`, and `n9` determine the II, which are on the critical path and colored in green. The DFG nodes `n10` and `n11` forming another recurrence cycle are colored in blue. The left DFG nodes that are not on any critical path are colored in grey. The scheduling will repeat every II (i.e., 4) cycles. n5 has to be mapped on a leftmost tile as it is a load operation and only the leftmost tiles are connected to the scratchpad memory. In addition, `tile0` has `n8` on it at `cycle1`. However, `tile0` is also used for receiving/routing the data at `cycle0` and `cycle3`. Note that we use the 4×4 CGRA as a motivating example for simplification, the proposed ICED prototype is based on a 6×6 CGRA.

One simple approach to maximize utilization is to increase the size of the DFG with loop unrolling. However, this is not always practical, because performance does not linearly increase with the unrolling factor due to loop-carried data dependencies. In fact, in our example, the II of `spmv` and `gemm` changes from 4 to 7 after unrolling by a factor of 2. Additionally, larger DFGs make the mapping process more complicated, potentially leading to sub-optimal IIs [16], [20]. The ICED approach to improving the utilization and energy efficiency is orthogonal to DFG transformations (such as unrolling) and the use of different mapping strategies. Actually, ICED could be used in combination with any of them.

B. Benefits of DVFS

Figure 3 shows how CGRA utilization and energy efficiency can benefit from the reduced voltage and frequency. Note that the conventional mapping strategy aims to minimize the II for

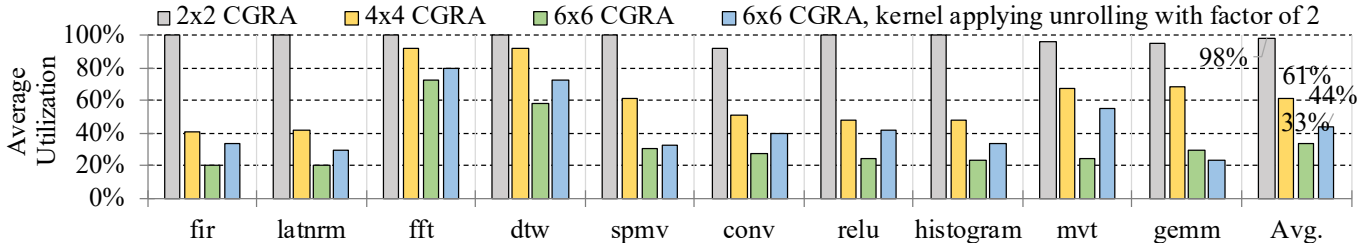
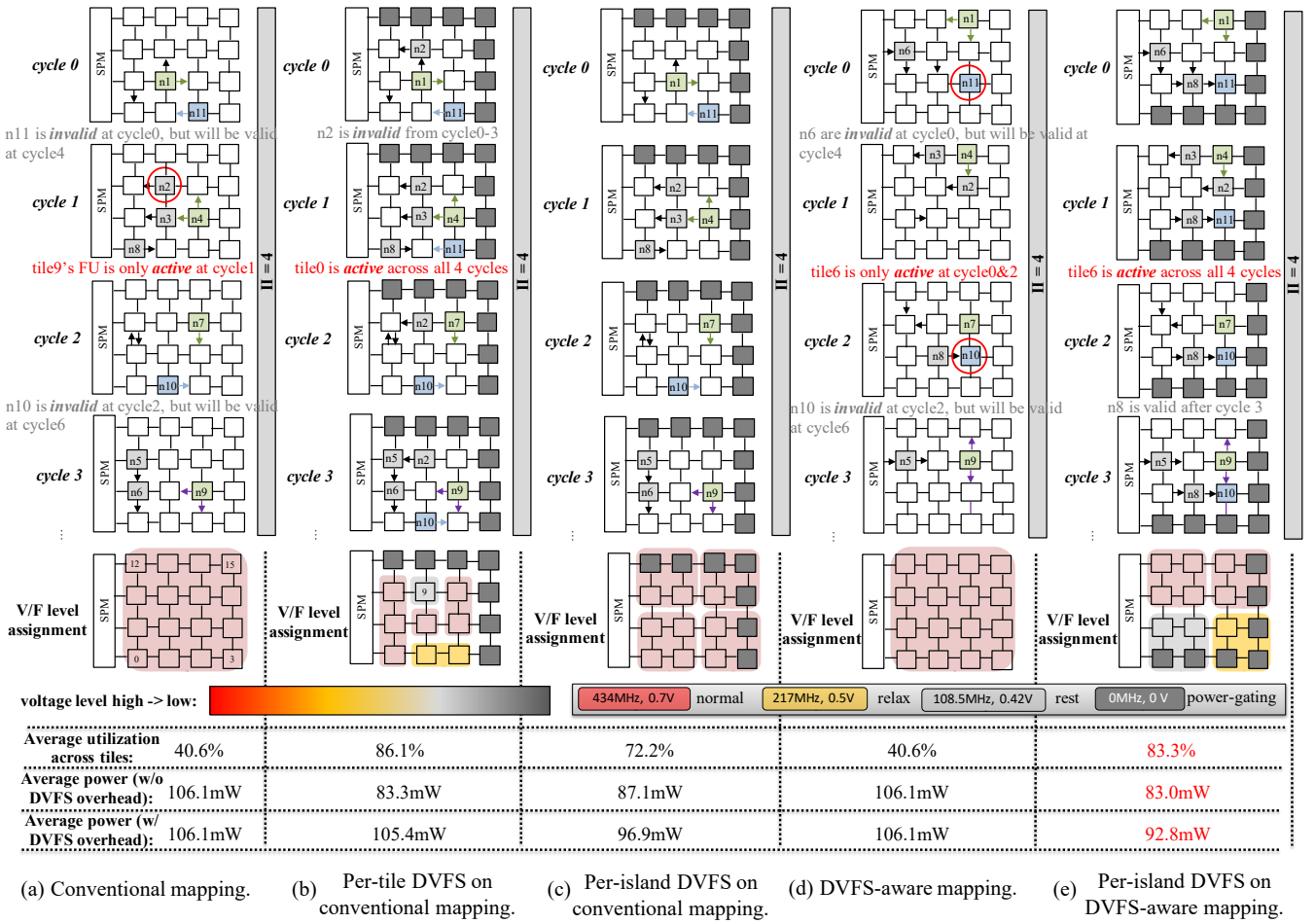


Fig. 2: Under-utilization across all kernels – the II of `spmv` and `gemm` changes from 4 to 7 after unrolling by a factor of 2.



(a) Conventional mapping. (b) Per-tile DVFS on conventional mapping. (c) Per-island DVFS on conventional mapping. (d) DVFS-aware mapping. (e) Per-island DVFS on DVFS-aware mapping.

Fig. 3: Motivating example for DVFS-aware co-design – A synthetic kernel mapped onto different CGRAs with different mapping strategies. (a) shows how the kernel is mapped on a conventional CGRA without DVFS support. (b) illustrates how the kernel is mapped on a CGRA with per-tile DVFS support, which introduces additional overhead due to the DVFS but benefits from the saved power thanks to the lowered V/F and power-gating. (c) has lower DVFS overhead as its support is on per-island basis instead of per-tile. However, it misses the opportunities to lower the frequency to `relax` (half of `normal`) and `rest` (a quarter of `normal`) for some tiles as all the tiles in one island are controlled together. (d) shows how the kernel is mapped onto the CGRA using our proposed DVFS-awared mapping. (e) illustrates the benefits when the per-island DVFS is enabled based on the mapping of (d). Its utilization is almost same as the per-tile DVFS and has the lowest overall power consumption.

higher performance but might sacrifice the tile utilization. The last row of the figure shows the DVFS modes for each tile of the CGRA.

As shown in Figure 3(a), without DVFS, the `n2` operation only occupies `tile9` at `cycle1`, indicating only 20% utilization of its functional units. If, as shown in Figure 3(b), we reduce the frequency of `tile9` to 1/4 of the original frequency leveraging per-tile DVFS, it will complete its com-

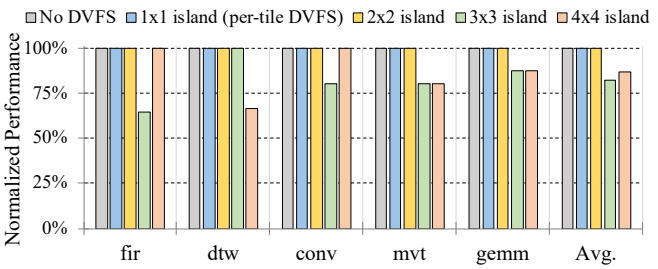


Fig. 4: Motivating example for island size – Normalized performance w.r.t. conventional CGRA (no DVFS) on an 8×8 CGRA with different DVFS island sizes. 2×2 DVFS island has no performance degradation compared with the CGRA without DVFS and the CGRA with per-tile DVFS support. Note that there would be irregular island shape for the 3×3 case on the 8×8 CGRA. We use the 8×8 CGRA to evaluate the performance across different island sizes, the proposed ICED prototype is based on a 6×6 CGRA.

putation and communication in 4 clock cycles with respect to the base clock, increasing utilization (from 40.6% to 86.1%) and reducing energy consumption (of 56%). ICED provides 3 different DVFS levels (in addition to the power-gated state): `normal`, `relax`, and `rest`. The frequency of `relax` level is half of `normal` level while the frequency of `rest` level is half of `relax` level and, thus, a quarter of `normal` level.

$$freq_{normal} = 2 \times freq_{relax} = 4 \times freq_{rest} \quad (1)$$

Our energy estimates are based on equation 4. All the detailed power statistics about tile, DVFS, scratchpad memory, and the correlation between voltage and frequency are discussed in Section V-A.

$$P(tile) = C \times V(tile)^2 \times freq(tile) + P_{static}(tile) \quad (2)$$

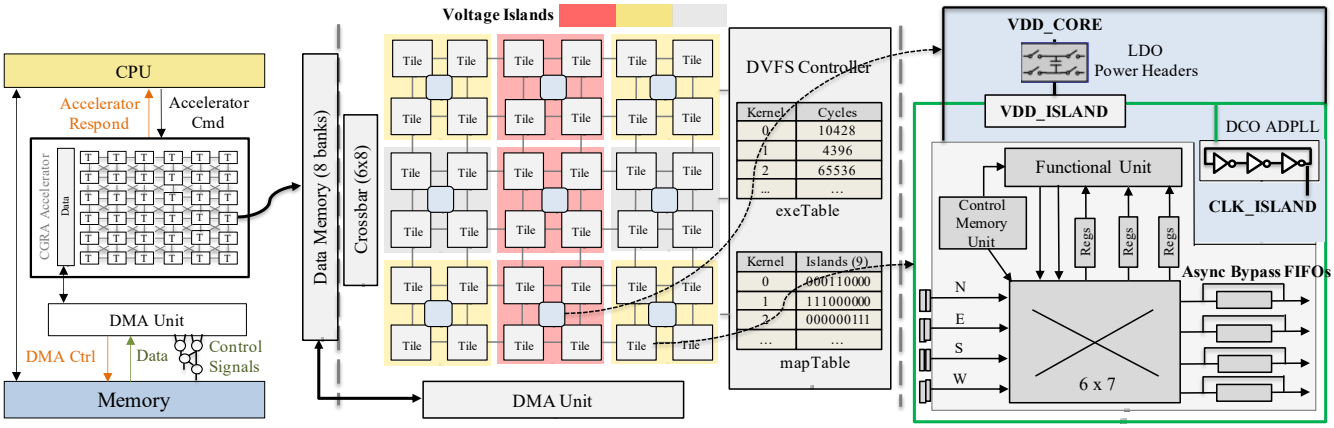
$$P_{non\_tile} = P_{SRAM} + \sum P_{DVFS\_overhead} \quad (3)$$

$$Energy = (\sum_{tile} P(tile) + P_{non\_tile}) \times ExecTime \quad (4)$$

It is important to highlight that tiles whose frequency can be lowered need to be identified considering the data dependency of the operations they are executing to guarantee correctness of the results. For example, while Figure 3(b) shows that `n5` starts at `cycle3` before `n2` completes, `n5` is actually considered invalid due to the predication-based data-flow. `n5` is first considered valid at `cycle7`, guaranteeing that the data dependency from `n2` to `n5` is respected. In addition, the routing signals in the tile's crossbar should not conflict in the enlarged period. Figure 3(a) shows that data movements required by `n2` (mapped to `tile9`), from `tile5` (south) to `tile9` (north) and from `tile9` (east) to `tile8` (west) happen only once in the 4-cycle II, making it possible to reduce the frequency of `tile9` to a quarter of the base frequency (i.e., the `rest` DVFS level). At the opposite, although `tile0` executes only one operation (multiplication `n8`) during the 4 cycles of the II, it requires two data movements for the two inputs, happening at `cycle0` and `cycle3` respectively. This prevents the reduction of the frequency (and voltage) of `tile0`, as shown in the map in the last row of Figure 3(b). Moreover, it is best to not reduce the frequency of the tiles that execute nodes in the critical path of the DFG (i.e., the nodes of the DFG that determine the II) to not decrease performance. For example, the tiles executing DFG nodes `n1`, `n4`, `n7`, and `n9` are always at the `normal` DVFS level and labeled with the green color across all the mappings in Figure 3.

Per-island DVFS – In the ICED framework, we propose to support DVFS on CGRAs leveraging power islands. While with a per-tile approach, each tile needs a DVFS Controller, with a per-island approach we only need one DVFS Controller for a group of contiguous (an island) tiles, leading to lower DVFS-related overhead (see Figure 8). Figure 3(c) shows what happens if a per-island DVFS approach, where each island is a 2×2 group of tiles, is applied to the conventional mapping from Figure 3(a) and used for the per-tile DVFS approach of Figure 3(b). Unfortunately, it is not possible to reduce the frequency of any island, since the DFG nodes on the critical path (i.e., `n1`, `n4`, `n7`, and `n9`) are all onto the 4 different islands. Therefore, to efficiently implement a DVFS approach based on power islands for CGRAs, it is necessary to support a DVFS-aware mapping process. Figure 3(d) shows the mapping when the different power islands are taken into consideration. We can see that the nodes on the DFG critical path are mapped in the same island. After independently managing the frequency of each island, the final mapping with DVFS enabled can be seen in Figure 3(e), which achieves 1.14× improvement in terms of power consumption over the baseline.

Size of the DVFS islands – Supporting DVFS on a per-tile basis, as proposed in UE-CGRA [35]), introduces overheads for both power and area (more than 30% of a tile) of the controller. With ICED, instead, we propose a framework able to generate CGRAs and map applications considering a per-island DVFS approach. To make such an approach effective, the framework itself should allow for careful determination of the size of the DVFS islands. Figure 4 shows the normalized performance of a set of kernels running on 8×8 CGRA with DVFS islands of varying sizes with respect to a per-tile DVFS approach. We can see that for this set of applications, there is no performance degradation only for the 2×2 DVFS island configuration, while the other configurations all are slower. This means that as the size of the islands increases, the II of the mapped kernels becomes longer. There are two reasons for this behavior. First, the II of the selected kernels (see Table I) usually is longer than 3 cycles. This allows us to easily map the kernels temporally in one or more 2×2 islands. Second, larger DVFS islands significantly limit mapping opportunities. For example, if a 4×4 island executes at the `rest` DVFS level, then the mapper will not assign nodes and edges in the DFG critical path to any tile of the island (16 tiles). This will in turn reduce resources available to the mapper and lead to sub-optimal results with low performance. For best



(a) CGRA SoC (b) Architecture of ICED CGRA (c) ICED one tile within a voltage island

Fig. 5: ICED architecture – DVFS islands are highlighted in different colors. Each voltage island can be set to a voltage level at runtime. Red indicates `normal level`, yellow indicates `relax level`, and grey indicates `rest level`.

energy-efficiency without reducing performance, ICED selects a 2×2 DVFS island size. Note that, in practice, DVFS island size is a design parameter that can be optimized specifically. In addition, ICED compiler can take in any island size for compilation and DVFS co-design.

Use Case for Streaming Applications – Besides improving the energy efficiency of a CGRA when executing single application kernels, DVFS can also be successfully applied to a CGRA when executing multi-kernel data-dependent streaming applications. For example, in an application like classifying a stream of data with a graph convolutional neural network (GCN), the execution time of the sparse matrix-vector multiplication (SpMV) kernel varies with the number of non-zeros in the input graphs (typically represented using sparse data formats, such as the compressed sparse row - CSR -, or the coordinate list - COO - format) [29], while the other stages (such as the weights combine kernel of Table I), always have a fixed execution delay independent from variations of the input data. Therefore, the kernel that limits the performance of the application (bottleneck) shifts during execution, depending on the characteristics of each input instance. The consequence is a significant load imbalance of the tiles allocated to different kernels as the application executes. Providing a large amount of resources to allow the SpMV kernel to complete quickly does not increase the overall throughput when the input graph is sparse and the weights combined kernel becomes the bottleneck. At the opposite, with a denser graph, the SpMV might become the bottleneck, with the weights combined kernel having to idle waiting for the data. Thus, lowering the voltage and frequency level of the tiles that are allocated to the non-bottleneck kernels through DVFS might allow to achieve higher energy efficiency for this class of applications (the architecture support is detailed in Section III-B while the compiler support is in Section IV-B).

III. ICED Architecture

This section introduces the ICED architecture and explains how it supports DVFS for kernels and streaming applications.

Figure 5 provides a high-level overview of the ICED architecture. ICED's CGRA is loosely coupled with a central processing unit (CPU) and is invoked through the accelerator command interface. A direct memory access (DMA) unit loads data and control signals into the CGRA's data scratchpad memory (SPM) and into the CGRA's control memory, respectively. Figure 5 (a) shows an example of such SoC with a 6×6 tile ICED CGRA. A typical CGRA tile contains a set of registers, a configuration memory containing the control signals, a set of functional units (FUs), and a 6×7 crossbar and bypass buffers that routes data internally or from/to other tiles. Our prototype has a 32KB SPM with eight banks. Each bank has a pair of read and write ports. The CGRA accesses the SPM via a 6×8 crossbar. The compiler needs to guarantee that the data required by the target kernel and application can fit (e.g., using tiling) into the 32KB SPM. There are 9 DVFS islands and each of them covers 4 tiles via a DVFS control unit (Figure 5(c)). A DVFS Controller (right side of Figure 5(b)) can control all the DVFS islands through the control units.

A. ICED DVFS Support

The ICED CGRA design clusters tiles into different voltage islands to enable per-island DVFS (Figure 5 (b)). Each voltage island is integrated with an on-chip standard power header cells based linear voltage regulator (LDO) [2], an all-digital PLL (ADPLL) [1], [28], and a DVFS control unit. The frequency/voltage sweep coordinates for ADPLL are stored in the look-up table, and the optimal voltage level (VDD_{ISLAND}) for each island will be determined according to the tile utilization during the application kernel mapping. Due to the clock domain crossing between different voltage islands, asynchronous buffers are required at ICED tile's data interfaces. Note that, CGRA tiles typically implement bypass buffers for data routing and can be re-designed for clock

domain crossing, which requires another clock port. In our prototype, we support 3 different DVFS levels and power-gating. The details about the correlation between voltage and frequency for those 3 levels and the corresponding power and area overheads are discussed in Section V.

B. ICED Streaming Applications Support

When executing a single kernel using the entire CGRA, voltage/frequency (V/F) levels are determined at the compilation time. While when a streaming application containing multiple kernels is accelerated, the voltage/frequency levels of each kernel is dynamically configured based on the execution status to achieve greater energy efficiency.

ICED's integrated voltage regulator and all-digital phase-locked loop (ADPLL) dynamically switches per-island voltage and frequency after collecting the execution information during a time window. Similar to DRIPS [29], ICED uses an input interval of 10 executions (10-round time window, i.e., once the 10th input is consumed) to trigger the dynamic switch of voltage and frequency via the DVFS Controller. The DVFS Controller maintains a `exeTable` and a `mapTable`. Once a kernel completes its execution, the termination signal will be sent to the DVFS Controller, and the `exeTable` is updated. After 10 updates, the bottleneck/non-bottleneck kernels are identified and the DVFS changes are triggered on the corresponding islands (based on the `mapTable`). The overhead of the DVFS Controller is trivial compared with the entire CGRA fabric. When a very large-scale CGRA is modeled and there are many kernels, distributing the DVFS Controller can facilitate the routing of the termination signal.

For instance, the GCN streaming application changes its DVFS level after every 10 graphs. The execution time of each kernel is collected by the DVFS Controller, and used to identify bottleneck and non-bottleneck kernels. After each time window, the voltage/frequency level of islands allocated to the bottleneck kernels is increased by one level while the voltage/frequency level of islands allocated to the non-bottleneck kernels are decreased by one level, if possible. We adopt ns-scale voltage regulator (as shown in Section V-A for the integrated voltage regulator) while the time needed to capture the execution status of a 10-round time window is in ms. How to allocate islands to each pipeline stage is detailed in Section IV-B.

IV. ICED Compiler Toolchain

ICED proposes an integrated compiler toolchain to facilitate the application and kernel mapping on the DVFS-enabled CGRA for energy-efficient acceleration. The toolchain is roughly divided in two parts: one responsible for the application mapping, the second for the kernel mapping. In this section, we first present how a kernel is mapped on CGRA resources (Figure 6) considering the available DVFS functionalities, and then show how an entire given application composed of multiple kernels is mapped on the entire CGRA fabric (Figure 7).

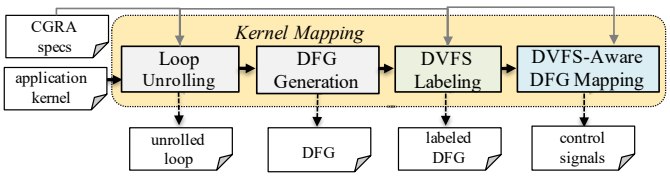


Fig. 6: ICED kernel mapping – The kernel mapping is a part of the integrated toolchain as shown in Figure 7.

The ICED compiler represents the application kernel as a DFG, where nodes are operations and edges indicate data dependencies between operations. However, to also consider DVFS requirements, each node of the DFG is initially labeled with a preferred DVFS level (i.e., i.e., `normal`, `relax`, or `rest`) before the mapping process onto the ICED CGRA starts. Equation 1 shows the relationship between different DVFS levels.

Loop Unrolling – A loop body that only has a few operations (i.e., with a small DFG) might lead to a mapping that under-utilizes the CGRA's computing resources, reaching sub-optimal performance. Unrolling the loop increases the number of operations but also leads to a longer II and complicates the mapping. We consider the effects of different unrolling factors on CGRAs of different sizes in our evaluation.

DFG Generation – Each DFG node represents an LLVM instruction, executed by a specific FU of our CGRA architecture. Control dependencies are converted to data dependencies using partial predication [12]. Data dependencies are represented by the edges of the DFG.

A. Kernel Mapping

DVFS Labeling – The conventional DFG mapping for CGRA treats every DFG node in the same way to obtain a performance-optimal solution. However, this is not enough for a CGRA that supports DVFS. As shown in Figure 1, the performance of a kernel mapped on a CGRA with enough resources is usually constrained by the maximum recurrence data dependency (i.e., RecMII). As previously discussed, this implies that running nodes of the DFG on the critical path at higher DVFS levels than other nodes provides better performance. To address this requirement, we propose a DVFS labeling algorithm that assigns to each DFG node a preferred DVFS level before mapping it to the ICED CGRA. Note that we also use the recurrence cycle to represent the recurrence data-dependency.

As shown in Algorithm 1, our approach labels the nodes on the longest recurrence cycles with the highest DVFS level, `normal` (line 17). If the length of a recurrence cycle is no more than half of the longest one, the nodes on it are labeled with the DVFS level of `relax` (i.e., 50% of `normal`, as shown in line 11). The algorithm labels the DVFS level of all the other DFG nodes as `rest` (line 25) or `relax` (line 28) considering the number of available CGRA tiles across the time domain (i.e., II cycles). For example, in the synthetic kernel shown in Figure 1, after labeling the 4 green DFG

Algorithm 1: LabelDVFSLevel

```
1  Input: DFG, targetCGRA, II
2  Result: DFG with DVFS level labeled
3  unlabeledNodes = CollectNodesAsSet(DFG);
4  recurrenceCycles = GetRecurrenceCycles(DFG);
5  longestCycle = GetLongestCycle(recurrenceCycles);
6  normalNodes = 0, relaxNodes = 0, restNodes = 0;
7  foreach cycle of recurrenceCycles do
8      if cycle.size() ≤ longestCycle.size() / 2 then
9          foreach dfgNode of cycle do
10             if dfgNode is in unlabeledNodes then
11                 LabelDVFS(dfgNode, relax);
12                 relaxNodes += 1;
13                 unlabeledNodes.Remove(dfgNode);
14     else
15         foreach dfgNode of cycle do
16             if dfgNode is in unlabeledNodes then
17                 LabelDVFS(dfgNode, normal);
18                 normalNodes += 1;
19                 unlabeledNodes.Remove(dfgNode);
20 foreach dfgNode of unlabeledNodes do
21     normalTiles = GetAvaiableTilesXII(normalNodes,
         normal, II);
22     relaxTiles = GetAvaiableTilesXII(relaxNodes, relax, II);
23     restTiles = GetAvaiableTilesXII(restNodes, rest, II);
24     if restTiles > 0 then
25         LabelDVFS(dfgNode, rest);
26         restNodes += 1;
27     else if relaxTiles > 0 then
28         LabelDVFS(dfgNode, relax);
29         relaxNodes += 1;
30     else
31         LabelDVFS(dfgNode, normal);
32         normalNodes += 1;
```

Algorithm 2: DVFS-Aware Mapping

```
1  Input: DFG, targetCGRA
2  Result: Mapping of DFG on MRRG of targetCGRA
3  orderedNodes = TopologicalOrder(DFG);
4  RecMII = AnalyzeRecurrenceEdges(DFG);
5  ResMII = (#Nodes in DFG)÷(#Tiles in targetCGRA);
6  II = Max(RecMII, ResMII);
7  DFG = LabelDVFSLevel(DFG, targetCGRA, II);
8  while Mapping is not available do
9      MRRG = CreateMRRG(targetCGRA, II);
10     foreach node of orderedNodes do
11         foreach Unmapped tile of MRRG do
12             labeledDVFS = GetLabeledDVFS(node);
13             island = GetDVFSIsland(tile);
14             assignedDVFS = GetAssignedDVFS(island);
15             if assignedDVFS == NULL then
16                 assignedDVFS = labeledDVFS;
17             if labeledDVFS ≤ assignedDVFS then
18                 tileCost = CalculateCost(node, tile,
                     assignedDVFS);
19                 tilesWithCostAndDVFS[tileCost] =
                     assignedDVFS;
20         optimalTileCost = Min(tilesWithCostAndDVFS);
21         dvfsLevel = tilesWithCost[optimalTileCost];
22         island = GetDVFSIsland(optimalTileCost.tile);
23         foreach tile in island do
24             AssignDVFS(tile, dvfsLevel);
25         ScheduleAndRoute(node, optimalTileCost,
             dvfsLevel);
26     II = II + 1;
```

nodes on the longest recurrence cycle with `normal` and the 2 blue nodes with `relax`, there are 5 DFG nodes (in grey color) left. Meanwhile, there are 8 CGRA tiles (i.e., two 2×2 DVFS-islands) that can execute at `rest` DVFS level assuming a 4×4 CGRA with a DVFS island of 2×2 tiles targeting an II of 4 cycles, which allows the left 5 grey DFG nodes to be labeled with `rest` DVFS level.

If there still are DFG nodes not yet labeled with a DVFS level before allocating all the CGRA tiles across the time domain, the algorithm labels the remaining DFG nodes with the highest DVFS level (`normal`, (line 31)) to not risk to reduce performance. In fact, DFG nodes that can execute at lower DVFS levels (i.e., `relax` or `rest`) could occupy a CGRA tile longer (i.e., 2× or 4×) than DFG nodes at the `normal` DVFS level. This would, in turn, limit placing and routing options during the DFG mapping and lead to a sub-optimal mapping with lower performance (i.e., lower II). It is important to note that the DVFS levels are not actually assigned at this stage of the mapping process. The DVFS labeling guides the DFG mapping, but the final DVFS level of each DFG node can still be adjusted by the heuristic mapping algorithm, as detailed in the following subsection.

DVFS-Aware DFG Mapping – We implement a heuristic DVFS-aware algorithm (Algorithm 2) that maps the DFG onto the CGRA's Modulo Routing Resource Graph (MRRG, line 9) [22] with the objective of minimizing the II. The MRRG is a time-space representation of the CGRA architecture. The algorithm iteratively increases the II (line 26) until it finds a valid mapping between the DFG and the available hardware resources. The mapping process applies Dijkstra's algorithm to find the shortest path between tiles and route data communication between operations (line 18).

The algorithm computes a cost whenever a DFG node labeled with a DVFS level is mapped onto a tile (line 18). The DVFS level labeled on a DFG node is determined by Algorithm 2. For example, a DFG node labeled with a DVFS level of `relax` has higher mapping priority (lower cost) onto a tile in a `relax` island over the other tiles (e.g., tiles in an island with a `normal` DVFS level). Our heuristic algorithm cannot map a DFG node labeled at a given DVFS level onto a tile with a lower DVFS level ((line 17) to reduce its complexity and the size of the search space.

When the algorithm maps a DFG node onto a specific CGRA tile, it takes into consideration the tile's routing capability to guarantee the feasibility of the communication and

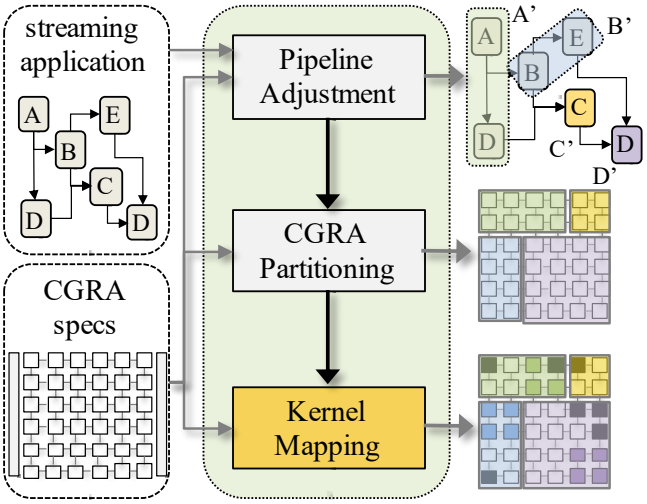


Fig. 7: ICED integrated compiler framework – The framework is implemented on top of the LLVM infrastructure [18]. Different colors in the bottom-right CGRA demonstrate that the islands allocated to each kernel are dynamically adjusted to different DVFS levels.

the assigned DVFS level (line 25). For example, considering the base clock, a DFG node executes only for 1 cycle on a tile set to `normal`, and for 4 cycles on a tile set at `rest`. In other words, the period of 1 `rest` cycle equals the period of 4 `normal` cycles from the point of view of the base clock. ICED currently targets a CGRA with single-cycle FU. The support for multi-cycle pipelined FUs (e.g., APEX [23]) can be easily integrated in ICED compiler and will provide even greater opportunities for ICED DVFS (e.g., an island with a mix of multi-cycle FUs and single cycle FUs).

The proposed ICED compiler toolchain uses a two-step approach to yield optimal solutions within tens of seconds on our evaluated kernels, which is similar to [16], [20] and much faster than the ILP-based mapping proposed in [4].

B. Application Mapping

By considering DVFS as one of the optimization parameters, ICED can also improve energy efficiency when the CGRA executes a pipeline of streaming data-dependent kernels. To enable energy-efficient acceleration of streaming applications, we extend the ICED compiler to enable the mapping of multiple interconnected kernels onto a single CGRA fabric, as shown in Figure 7.

Pipeline Adjustment – Similar to the approach presented in DRIPS [29], the ICED compiler can combine kernels together to meet constraints in terms of data memory size. In ICED, however, the kernels are mapped at the granularity of a DVFS island (e.g., a set of 2×2 tiles). Hence, the maximum number of kernels is limited to the number of islands supported by the specific ICED CGRA design. Sub-kernels merged in a single kernel are time-multiplexed at runtime on the tiles allocated for such a combined kernel.

CGRA Partitioning – Our approach partitions the CGRA across multiple kernels, but each kernel occupies at least one island. The actual number of tiles allocated to each kernel is determined by considering its average execution time in the context of the entire streaming application. To obtain the average execution time for each kernel, we randomly pick 50 input instances from the actual application datasets and execute the application in its entirety. Mapping a kernel onto a different number of islands leads to different speedups. We employ the kernel mapping algorithm detailed in Section IV-A but add an additional constraint (i.e., limited DVFS levels, as detailed in the next paragraphs). At this stage, we exhaustively evaluate the mapping for each kernel on a varying number of islands (each candidate indicates a specific partition). For each streaming application, the algorithm selects the CGRA partitioning that minimizes the overall application throughput. Such exhaustive exploration is performed offline, at compilation time, thus it does not affect the runtime performance. At runtime, the configuration of each kernel does not change, but the DVFS level of each island is dynamically adjusted to improve the energy-efficiency.

Kernel Mapping – Section IV-A details the kernel mapping algorithm. However, to support DVFS in streaming applications, we consider an additional constraint when mapping kernels. DFG nodes and tiles of a partition are allocated with either `normal` or `relax` DVFS level by the compiler. This is necessary because the DVFS levels of all the islands allocated to one kernel need to be adjusted in a synchronized manner, and `rest` is the lowest DVFS level in the current ICED prototype. During runtime, the non-bottleneck kernels are able to lower their DVFS levels, which might be increased back later if the kernels become bottlenecks again (the execution status is monitored by the DVFS Controller showing in Figure 5).

Note that ICED's approach can easily support more DVFS levels, adapting both the mapping algorithms and the CGRA design. We present our approach by considering three DVFS levels, plus the additional power-gated level, but it is easily parameterizable/customizable in our toolchain without any constraint on the number of DVFS levels.

V. Experimental Evaluation

This section details the experimental setup and results. We evaluate ICED's performance, utilization, scalability, and energy-efficiency, with kernels and streaming applications from different domains.

ICED modeling and Compiler – We model our ICED in RTL using PyMTL3 [13], [31], [33] and generate the synthesizable Verilog. We use Synopsys to perform logic synthesis and obtain the power, area, and timing of the design. We use CACTI 6.5 [34] to evaluate the power and area of the SRAM. Section V-A provides details of the synthesis statistics. ICED's compiler framework is built on top of the LLVM [18] infrastructures.

Benchmark Kernels – We evaluate ICED with kernels from three domains: embedded, ML, and HPC. Table I shows the characteristics of each kernel. For the embedded domain,

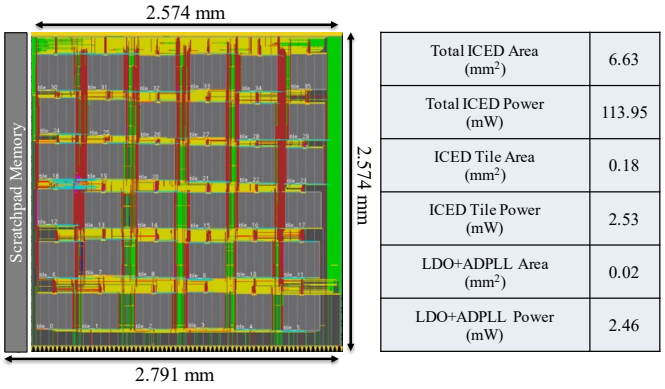
Domain/ Application	Kernel	Data	Unrolling Factor = 1			Unrolling Factor = 2			Target CGRA with Islands
			Nodes	Edges	RecMII	Nodes	Edges	RecMII	
Embedded domain	fir	64	12	16	4	20	26	4	n × n (2 × 2)
	latnrm	32	12	16	4	19	25	4	n × n (2 × 2)
	fft	1024	42	60	4	71	100	4	n × n (2 × 2)
	dtw	128^2	32	49	4	51	84	4	n × n (2 × 2)
Machine learning	spmv	512	19	24	4	37	50	7	n × n (2 × 2)
	conv	32^2	17	23	4	24	34	4	n × n (2 × 2)
	relu	1024	14	19	4	23	32	4	n × n (2 × 2)
High performance computing	histogram	2048	15	17	4	23	26	4	n × n (2 × 2)
	mvt	128^2	20	29	4	37	54	4	n × n (2 × 2)
	gemm	128^2	17	24	4	23	37	7	n × n (2 × 2)
2-layer Graph Convolutional Network (GCN)	Compress	ENZYME 600 graphs 450 for training 150 for inference	24	32	4	46	65	7	1 (2 × 2)
	Aggregate (x2)		27	34	4	53	69	7	4 (2 × 2)
	Combine		26	35	4	51	71	7	1 (2 × 2)
	CombRelu		30	42	4	59	85	7	2 (2 × 2)
	Pooling		16	21	4	31	43	7	1 (2 × 2)
Synthesized Lower–Upper (LU) Decomposition kernels	Init	150 matrices (within the size of 100×100) selected from the University of Florida sparse matrix collection	11	15	4	21	32	7	1 (2 × 2)
	Decompose		15	25	4	27	50	7	1 (2 × 2)
	Solver0		33	49	8	65	98	15	2 (2 × 2)
	Solver1		35	54	12	69	108	23	2 (2 × 2)
	Invert		14	22	4	24	37	4	1 (2 × 2)
	Determinant		20	36	7	38	71	13	2 (2 × 2)

TABLE I: Target workloads and streaming applications from different domains – The RecMII indicates the maximum recurrence cycle length derived from the inter-iteration data-dependency of each kernel. The kernels are mapped on and accelerated by the entire CGRA fabric. In contrast, each kernel of the streaming application only leverages one or multiple islands. There are 9 islands in total in a 6×6 ICED CGRA.

we use digital signal processing (DSP) kernels: finite impulse response filter (`fir`), normalized lattice filter (`latnrm`), fast-fourier transform (`fft`), and dynamic time warping (`dtw`). For the ML domain, we use: sparse matrix-vector multiplication (`spmv`), convolution (`conv`), and rectified linear unit (`relu`). Note that `relu` is usually fused with `gemm` or `conv` kernel for ML acceleration in practical. We use standalone `relu` kernel here to show that our approach is able to handle control flows. To represent the HPC domain, we chose: Histogram (`histogram`), matrix-vector product and transpose (`mvt`), and generalized matrix multiplication (`gemm`). The C/C++ implementations of the kernels are collected from the Poly-Bench [40], UTDSP, and Parboil [27] benchmarks.

Benchmark Streaming Applications – To demonstrate the advantages of DVFS, we evaluate ICED with two representative data-dependent streaming applications (comprising a total of 11 unique kernels) on ICED. The first application is a 2-layer `GCN` model derived from Pytorch-Geometric [8] and used to classify enzymes. The model is pre-trained and we only accelerate inference on the CGRA. The different sizes and structures of the enzymes, represented as graphs (edge degree from 2 to 126 with an average of 32.6 [3]), lead to an imbalanced pipeline. The GCN inference application is composed of 5 unique kernels, with one kernel (aggregate) repeated twice. The second application is a pipelined lower-upper decomposition (`LU`). It contains a total of 6 kernels organized in 4 pipeline stages because some of the kernels execute in parallel.

The kernels/applications we evaluated for ICED are from different application domains and have been previously used to evaluate state-of-the-art CGRA designs [10], [29], [37]. We aim to validate the generality and scalability of the proposed ICED hardware/software co-design approach.



Total ICED Area (mm^2)	6.63
Total ICED Power (mW)	113.95
ICED Tile Area (mm^2)	0.18
ICED Tile Power (mW)	2.53
LDO+ADPLL Area (mm^2)	0.02
LDO+ADPLL Power (mW)	2.46

Fig. 8: ASAP 7nm physical layout, and area and power (@ 0.7V/434MHz) breakdown of an ICED 6×6 CGRA.

Evaluation on Kernels and Streaming Applications – In the single kernel evaluation, the ICED compiler determines the voltage and frequency level before launching the kernel itself and maintains them throughout the entire kernel execution. This allows evaluating the benefits of embedding knowledge of DVFS features across the whole ICED compiler and architecture stack. When executing the streaming applications, instead, we allow the ICED stack to dynamically adjust voltage and frequency at runtime.

Evaluated CGRA Designs – The baselines for our comparison are a conventional CGRA without DVFS support (**Baseline**), and a CGRA that supports DVFS on a per-tile basis with power-gating enabled (**Per-tile DVFS + Power-gating**).

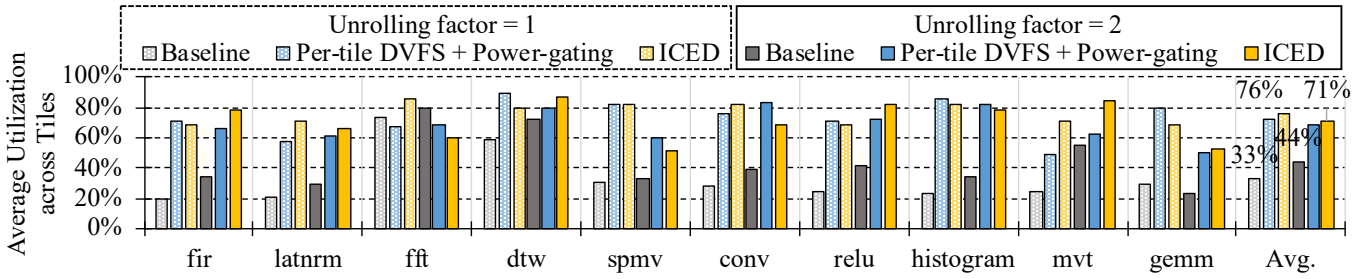


Fig. 9: Average utilization of tiles across different kernels – When the kernels are small, ICED can achieve even better utilization than per-tile DVFS (e.g., `fir`, `latnrm`, `mvt`) as the ICED mapping is more utilization-friendly thanks to the islandization.

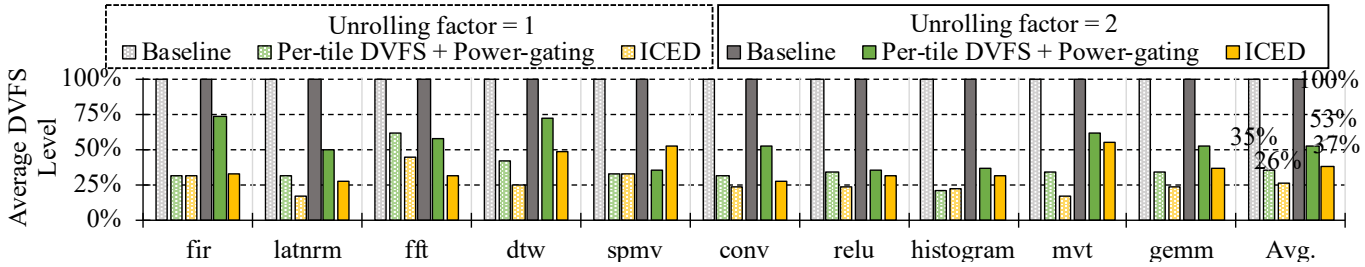


Fig. 10: Average DVFS level across different kernels – We treat `normal` as 100%, `relax` as 50%, `rest` as 25%, and power-gated as 0%, to calculate the average value.

ICED embeds per-island DVFS support in both compiler and architecture. The **Per-tile DVFS + Power-gating** is an improved version of UE-CGRA with spatio-temporal support, which can also be considered as a simplified ICED CGRA solution where each DVFS island is composed of only one tile. While, as discussed, the ICED framework allows to set of any size for the DVFS islands, considering the selected kernels, applications, and CGRA size, we adopt an island size of 2×2 tiles in this evaluation. In addition, we compare ICED to DRIPS [29], demonstrating how it can improve energy efficiency when accelerating streaming applications.

A. ICED Timing, Power, and Area

Figure 8 shows a placed-and-routed 6x6 ICED CGRA design from a bottom-up ASIC flow using the predictive ASAP 7nm FinFET cell library [5]. Without SRAM macros, this ICED design occupies 6.63 mm^2 while consuming an average power of 113.95 mW at its `nominal` VDD (0.7V) and clock frequency (434MHz).

ICED Voltage Island and DVFS Overheads – Our 6x6 ICED CGRA is partitioned into 9 voltage islands. To support DVFS, each ICED voltage island equips the all-synthesizable LDO and ADPLL from the FASoC open-source SoC design framework [1], [2]. Figure 8 includes the area and power breakdown for ICED tiles and the DVFS support.

ICED DVFS Mode – The impact of voltage scaling on the circuit propagation delay can be found via SPICE simulations [24]. When the supply voltage is reduced, the circuit propagation delay will increase, so clock frequency has to decrease to ensure no timing violations. Thus, V/F are often scaled as a pair for the best energy efficiency. For ICED, we explore the benefits of scaling the voltage of each island to the following levels: (1) `normal` mode (@ nominal 0.7V and 434MHz), (2) `relax` mode (@ 0.5V and 217MHz), (3) `rest` mode (@ 0.42V and 108MHz), and (4) `power_gating` mode gates the whole voltage island.

The choice of the above V/F levels is co-designed with ICED compiler, although the LDO and ADPLL we adopted are capable of ns-scale fine-grained on-chip DVFS if needed.

SRAMs for ICED – We employ CACTI 6.5 to obtain power and area of ICED's SRAM, considering a 22nm technology node. Our design uses a 32KB memory with eight banks. Each bank has a dedicated port. The entire SRAM occupies 0.559 mm^2 and consumes up to 62.653 mW of power. We highlight that CACTI 6.5 does not support the 7nm technology node. Thus our evaluation is conservative and we believe that our proposed solution would provide even better results in terms of power efficiency if the SRAM could be evaluated at 7nm (in 7nm, the baseline SRAM power and area will be largely reduced, hence our DVFS-enabled CGRA will achieve even greater total chip power savings).

B. Experimental Results

The evaluation is based on a cycle-accurate simulation according to the kernel mapping. The power and area for the CGRA is obtained from the post-layout implementation (Figure 8).

Performance and Utilization – Thanks to the proposed DVFS-aware DFG mapping, ICED achieves higher resource utilization without losing performance. The utilization is computed at each island according to its frequency. There are many idle tiles along the non-critical paths. Lowering the frequency

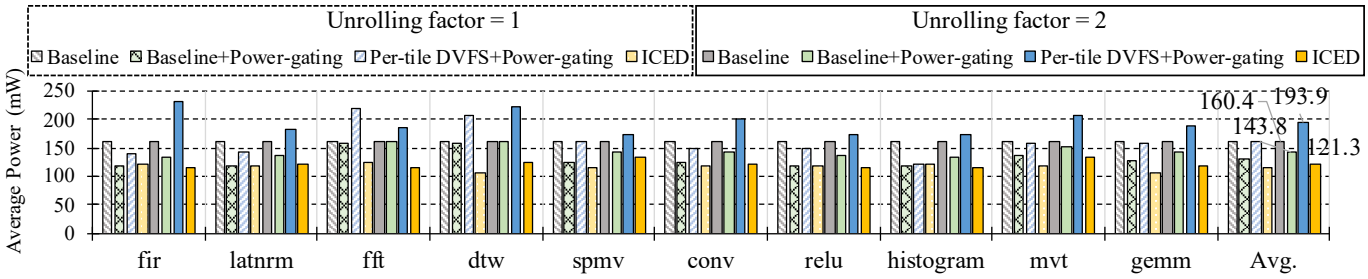


Fig. 11: Evaluation of energy-efficiency – As the performance (total execution time) stays the same for baseline, per-tile DVFS with power-gating, and ICED, the power consumption can be used to evaluate the energy-efficiency.

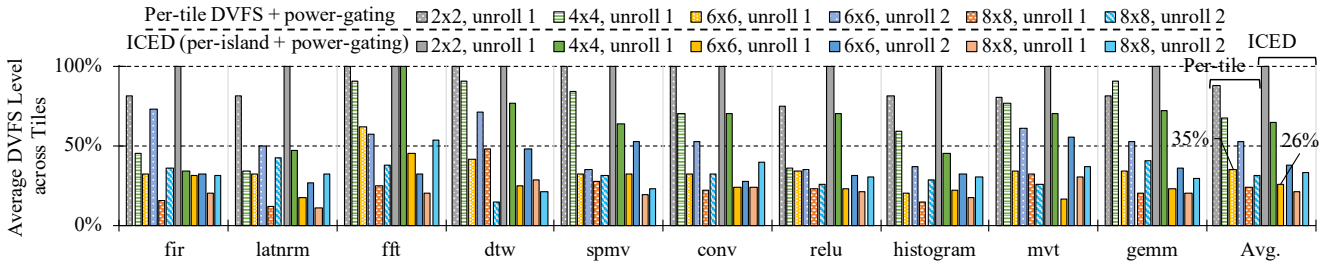


Fig. 12: Evaluation of scalability – ICED can achieve a similar average DVFS level value compared with the per-tile solution, especially when the relatively smaller kernel runs on larger CGRA.

of idle tiles (e.g., 434MHz to 108.5Mhz) is equivalent to higher utilization (4×) and lower power/energy without losing overall performance. ICED does not rely on the average utilization to perform optimization, instead, we look at opportunities that can improve the energy efficiency without hurting performance via DVFS. When doing so, the average utilization of tiles also improves. Figure 4 in Section II shows that for our set of kernels using a 2×2 island does not reduce the performance with respect to the no-DVFS baseline and the per-tile DVFS-aware CGRA design. Figure 9, shows that the average utilization across tiles increases from 33% to 76% (2.3×) and from 44% to 71% (1.6×) without and with unrolling, respectively. When the kernels are small, ICED can achieve even better utilization than per-tile DVFS (e.g., `fir`, `latnrm`, `mvt`). This happens because ICED prefers to map the dependent DFG nodes within an island, leading to higher utilization for certain tiles and more tiles in power-gating mode. We can also see that the utilization drops after applying unrolling. This happens for two reasons: the unrolled DFG becomes larger, complicating the mapping process (i.e., `fft`), and the II increases because of the unrolling itself (i.e., `spmv` and `gemm`).

Energy-Efficiency – Figure 10 and Figure 11 show the average DVFS level across tiles and power consumption of ICED, respectively. We consider `normal` as 100%, `relax` as 50%, `rest` as 25%, and power-gating as 0%, to calculate the average values. We can see that ICED is better than the per-tile DVFS solution, i.e., 35% vs. 26% and 53% vs. 37% without and with unrolling, respectively. The reason is that the naive per-tile mapping does not consider utilization. This means that the mapper might assign two dependent DFG nodes onto two tiles that are far away from each other as long as the II is not violated due to data routing. Instead, ICED tries to map the dependent DFG nodes close to each other considering the separate islands. Since the performance (total execution time) stays the same for the no-DVFS baseline, the per-tile DVFS with power-gating, and the ICED solution, we can just consider power consumption to evaluate improvements in energy-efficiency. With an unrolling factor of 1, the per-tile DVFS with power-gating consumes similar average power as the no-DVFS baseline. This happens because there actually is a power overhead for supporting DVFS on each tile. ICED, instead, experiences less overhead by supporting islandization. As shown in Figure 11, ICED consumes 121.3 mW average power while the baseline and the per-tile DVFS with power-gating CGRA consume 160.4mW and 193.9mW average power, leading to 1.32× and 1.6× energy-efficiency, repectively, when the unrolling factor is 2. Power-gating is also enabled in baseline for this evaluation, which shows average 1.12× energy-efficiency improvement (i.e., 160.4mW vs. 143.8mW). Note that power-gating benefits more on smaller DFGs regardless of II as there would be more tiles in idle.

Scalability – To demonstrate the scalability of our approach, we compare the average DVFS levels obtained while enabling per-tile DVFS and 2×@ DVFS island using the ICED framework to generate CGRAs of different sizes ((i.e., 2×2, 4×4, 6×6, and 8×8). Figure 12 shows that the ICED solution with islandization can achieve a similar average DVFS level value in comparison to the per-tile solution, especially when the relatively small kernel executes on large CGRA. For example,

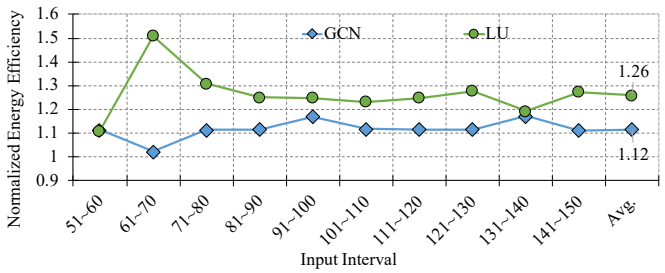


Fig. 13: Normalized energy efficiency (performance-per-watt, ICED over DRIPS) on GCN and LU Applications – The first 50 input instances are used to profile the initial mapping for DRIPS and ICED.

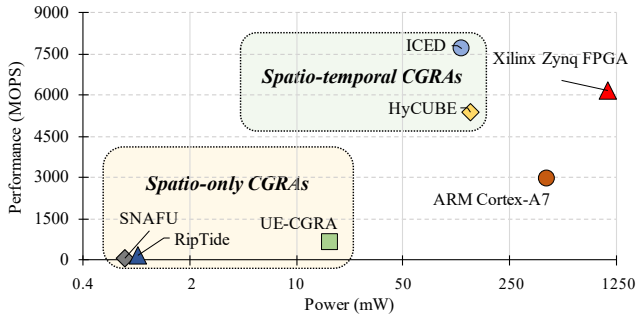


Fig. 14: Comparing ICED with other architectures – The data except ICED is derived from HyCUBE [37] and RipTide [10].

we see average DVFS levels of 35% vs. 26% on a 6×6 CGRA without unrolling.

Streaming Applications – We illustrate how ICED improves energy efficiency when accelerating data-dependent streaming applications with DVFS. Figure 13 shows the normalized energy-efficiency of ICED with respect to DRIPS CGRA design on two exemplar streaming applications (GCN and LU) across the applicaiton input intervals. We observe that compared with DRIPS, ICED improves energy-efficiency by an average of 1.12× for the GCN application and by 1.26× for the LU application, respectively. In this evaluation, we allow DVFS on ICED for every 10 input intervals (i.e., in ms), to ensure a fair comparison with DRIPS. However, ICED equipped with ns-scale voltage regulator allows finer-grain DVFS to achieve greater energy efficiency. DRIPS is a CGRA that specifically optimizes performance, while ICED optimizes power consumption without sacrificing performance. In fact, ICED can be applied together with DRIPS to enable greater energy efficiency.

Comparison with Other Architectures – Figure 14 shows the power and performance (derived from HyCUBE [37] and RipTide [10]) for running `FFT` kernels on various architectures. Note that it is hard to provide a fair comparison due to different technologies, number of tiles, specialized hardware features (e.g., single-cycle-multi-hop in HyCUBE, `stream` in RipTide), software optimizations (e.g., vectorization in SNAFU), and memory sizes/hierarchies, across platforms. Nevertheless, ICED's hardware/software co-design approach can be applied to any baseline CGRA to improve the power-efficiency.

VI. Related Work

Kernel Mapping – CGRAs accelerate standalone kernels by spatially and temporally mapping the operations of its data-flow graphs (DFGs) onto intercommunicating tiles composed of one or more FUs with respect to the data dependencies between the DFG nodes. [4] applies Integer Linear Programming (ILP) into mapping and yet suffers from long compilation time due to enormous search space. Several recent approaches [16], [20] use Machine Learning (ML) to optimize the mapping. LISA [20] uses Graph Neural Network to generate labels to guide the mapping and MapZero [16] leverages Reinforcement Learning to decide placement and routing. However, both methods are more suitable for homogeneous CGRAs and need significant effort to support tile-level DVFS. In contrast, cost-function-based heuristic [6], [15] makes it easy to apply new features by customizing the cost function. However, none of the existing approaches takes voltage and frequency as reconfigurable parameters during mapping.

DFG nodes can be fused leveraging specialized hardware support to shorten the DFG paths. For example, SPU [7] supports single-cycle `join` and RipTide [10] provides `stream` instructions to avoid explicitly iterating the index of the streamed data array with a control-flow in the DFG. However, even with a shorter recurrence cycle in the DFG (e.g., 1 as RecMII), if the DFG size is larger than the tile count (e.g., 37 DFG nodes mapped onto 6x6 tiles, $\lceil 37 \div 36 \rceil$ as ResII), the achievable II will be at least 2 in the context of spatio-temporal CGRA, leading to low utilization. Similarly, HLS expects the II to be 1 for statically bounded loops but the achievable II is constrained due to limited hardware resources (e.g., BRAM ports). Also, ICED is not limited to statically bounded loops or any specific DFG transformation. In many cases (e.g., nested-loop, complex control-flow, inter-iteration data dependency, hardware resource limitation) the achievable II cannot be shortened. In those cases, ICED can be applied to improve the overall system utilization, achieving higher power-efficiency.

Application Mapping – CGRAs are also leveraged to accelerate multi-kernel streaming applications. Chordmap [19] leverages a static spatial-temporal kernel mapping strategy to improve the overall throughput. Similarly, [14], [26], [39] propose statically partitioning their CGRA tiles to a limited group of kernels. However, they assume the fixed execution time of each kernel without considering the impact of the input data. PPA [25] and ARENA [32] propose hardware monitors to check resource availability and trigger dynamic partitioning of the reconfigurable substrate, aiming at higher utilization. DRIPS [29] considers the execution time of the bottleneck kernel to determine the resources that need to be allocated to each kernel at runtime, improving the overall throughput. DRIPS outperforms DynPaC [30] through dynamic reshape

mechanism. However, none of these works takes into account DVFS for energy-efficient acceleration.

DVFS-Aware CGRA – UE-CGRA [35] applies per-tile fine-grained DVFS to enable energy-efficient execution for irregular loops and achieves 1.24-2.32× energy-efficiency across five kernels. However, UE-CGRA targets only the **spatial** CGRAs, a subset of **spatial-temporal** CGRA without extending the resource along the temporal domain. This lowers the mapping complexity but requires more computing and routing resources for big kernels (i.e., an 8×8 UE-CGRA accelerates kernels with only 10-20 DFG nodes and more than half of the tiles are always in idle). Also, its per-tile fine-grained DVFS does not require a sophisticated compiler (as each tile only supports one operation during the entire execution, which can be viewed as a straightforward per-operation DVFS) but significantly increases the area and power overhead (i.e., more than 30%) of the entire chip. In addition, UE-CGRA has no support for streaming applications.

In contrast, ICED targets spatio-temporal CGRAs, allowing time-multiplexing of different operations on the same tile on a per-cycle basis. This requires a sophisticated DVFS-aware compiler to coordinate different operations across multiple cycles within one DVFS-island without lowering the performance for both standalone kernels and streaming applications. To the best of our knowledge, ICED is the first approach that addresses this problem. To make a fair comparison between UE-CGRA and ICED in the context of spatial-temporal CGRAs, we model a **per-tile DVFS + power-gating** architecture in our evaluation as an improved version of UE-CGRA with spatio-temporal support. The experimental results show that ICED outperforms it by 1.6× in terms of energy-efficiency.

Other Ways to Improve CGRA Utilization – DySER [11] allows speculative execution and squash to achieve higher utilization. SGMF [36] increases the buffer size for better utilization in the presence of long latency operations. Both require additional hardware resources with higher power consumption. In contrast, ICED opportunistically throttles down tiles running non-critical operations to improve the power-efficiency with minimal hardware overhead.

VII. Conclusion

This paper proposes ICED, an integrated framework to generate and map kernels on CGRAs supporting DVFS. ICED supports power island of configurable size, and is able to accordingly map applications. ICED is the first solution that supports island based DVFS on spatio-temporal CGRAs. We evaluate ICED on a set of representative kernels from the embedded, ML, and HPC domains with islands of 2×2 tiles, and show that it improves utilization 2.3× and energy efficiency 1.32× over a baseline solution without DVFS. We also evaluate the ICED approach on two streaming applications, demonstrating how it significantly improves energy efficiency (up to 1.26×) by dynamically switching frequency and voltage at runtime over a state-of-the-art partially and dynamically reconfigurable solution.

Acknowledgments

This research is partially supported by the National Science Foundation under Grant 2403409, by the Adaptive Tunability for Synthesis and Control via Autonomous Learning on Edge (AT SCALE) Laboratory Directed Research and Development (LDRD) Initiative and the ASCR Project Compilers Frameworks and Hardware Generators to Support Innovative US Government Designs at Pacific Northwest National Laboratory (PNNL), and by the National Research Foundation, Singapore under its Competitive Research Programme Award NRF-CRP23-2019-0003.

Appendix

A. Abstract

This section summarizes the artifact evaluation for this work. First, we provide the check-list for this artifact. Next, we describe the directory structure for the code. Finally, the installation, experiment workflow, and evaluation illustrate how to use the artifact to reproduce results and extend the implementation.

B. Artifact check-list (meta-information)

- **Compilation:** LLVM 12.0.
- **Data set:** https://github.com/tancheng/CGRA-Bench.
- **Run-time environment:** At least Ubuntu 16.04 and Python 3.7, if not using docker.
- **Hardware:** Any machine that can run a docker environment.
- **Metrics:** Kernels mapping statistics, ICED CGRA utilization, DVFS level, energy-efficiency, and scalability.
- **Output:** Script would directly generate figures.
- **How much disk space required (approximately)?:** 16GB.
- **How much time is needed to prepare workflow (approximately)?:** 10 minutes.
- **How much time is needed to complete experiments (approximately)?:** 3 hours.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Open Source Initiative BSD 3-Clause License.
- **Archived (provide DOI)?:** Will be added.

C. Description

1) How to access: Code for ICED compiler can be accessed at https://github.com/tancheng/CGRA-Mapper/tree/utilization. RTL code for ICED CGRA can be accessed at https://github.com/tancheng/VectorCGRA. We will also publish the initial artifact on Zenodo.

2) Hardware dependencies: Any machine that can run docker.

3) Software dependencies: If not using docker, ICED requires Ubuntu 16.04↑, Python 3.7↑, PyMTL3, Synopsys Design Compiler, Cadence Innovus, and CACTI6.5.

4) Data sets: Benchmark can be accessed at https://github.com/tancheng/CGRA-Bench.

D. Installation

To facilitate the evaluation of ICED, we strongly recommend using the provided docker environment (https://hub.docker.com/r/cgra/iced). Alternatively, users can follow the instructions in https://github.com/tancheng/CGRA-Mapper/tree/utilization to setup and perform their own experiments.

E. Suggested Experiment workflow

For the ease of artifact evaluation, we provide a sample bash script that automates launching and analyzing experiments of Table I, Figure 9, Figure 10, Figure 11, and Figure 12, which represent the main contributions of ICED.

For the CGRA physical design (Figure 8), we provide the Verilog RTLs for ICED 6x6 CGRA. The RTLs are synthesized with Synopsys Design Compiler and placed&routed in Cadence Innovus, using ASAP7 PDKs. Due to the EDA license, we are not able to provide access to commercial EDA tools. As a solution, we provide the original power, area, and timing reports (with timestamps) and the final layout for the evaluation. Alternatively, the provided RTLs can be physically designed with other open-source tools (e.g., OpenRoad) at the evaluators' choice, but the exact numbers might change.

Figure 13 and Figure 14 are mainly derived from DRIPS [29], HyCUBE [37], and RipTide [10], which are out of the scope in this evaluation.

Below are the key steps:

- Pull docker:
  `docker pull cgra/iced:demo`
- Launch docker:
  `docker run -it cgra/iced:demo bash`
- Check RTLs and physical design logs:
  `vi Fig_8_CGRA_Physical_Design/README.md`
- Setup simulation environment:
  `source /WORK_REPO/venv/bin/activate`
- Locate the script:
  `cd /WORK_REPO/CGRA-Flow/CGRA-Mapper/test`
- Run the script:
  `./demo.sh`
- Figure out the `CONTAINER_ID`:
  `docker ps`
- Copy figures out of the docker to verify:
  `docker cp CONTAINER_ID:/WORK_REPO/CGRA-Flow/CGRA-Mapper/test/example ./.`
- Visualize the results: The figures corresponding to Figure 9, 10, 11, and 12 are located inside the target folder. Table I is saved in `cvs` format that can be viewed by any text editor or imported to any online latex table generator to view.

F. Evaluation and expected results

The bash script in Section E produces the figures illustrating the main contributions of ICED. We set a 3-minute threshold for the evaluation of each kernel under a given CGRA configuration to avoid hanging (due to the heuristic mapping algorithm).

G. Experiment customization

Check out https://github.com/tancheng/CGRA-Flow to customize a CGRA design in both RTL and compiler.

H. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae

References

[1] Tutu Ajayi, Sumanth Kamineni, Yaswanth K Cherivirala, Morteza Fayazi, Kyumin Kwon, Mehdi Saligane, Shourya Gupta, Chien-Hen Chen, Dennis Sylvester, David Blaauw, Ronald Dreslinski, Benton Calhoun, and David D. Wentzloff. An open-source framework for autonomous soc design with analog block generation. In *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, pages 141–146, 2020.

[2] Suyoung Bang, Wootaek Lim, Charles Augustine, Andres Malavasi, Muhammad Khellah, James Tschanz, and Vivek De. 25.1 a fully synthesizable distributed and scalable all-digital ldo in 10nm cmos. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 380–382, 2020.

[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[4] S Alexander Chin, Noriaki Sakamoto, Allan Rui, Jim Zhao, Jin Hee Kim, Yuko Hara-Azumi, and Jason Anderson. Cgra-me: A unified framework for cgra modelling and exploration. In *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*, pages 184–189. IEEE, 2017.

[5] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. Asap7: A 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

[6] Nathan Clark, Manjunath Kudlur, Hyunchul Park, Scott Mahlke, and Krisztian Flautner. Application-specific processing on a general-purpose core via transparent instruction set customization. In *37th international symposium on microarchitecture (MICRO-37'04)*, pages 30–40. IEEE, 2004.

[7] Vidushi Dadu, Jian Weng, Sihao Liu, and Tony Nowatzki. Towards general purpose acceleration by exploiting common data-dependence forms. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 924–939, 2019.

[8] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[9] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. Xilinx adaptive compute acceleration platform: Versaltm architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 84–93, 2019.

[10] Graham Gobieski, Souradip Ghosh, Marijn Heule, Todd Mowry, Tony Nowatzki, Nathan Beckmann, and Brandon Lucia. Riptide: A programmable, energy-minimal dataflow compiler and architecture. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 546–564. IEEE, 2022.

[11] V. Govindaraju et al. Dyser: Unifying functionality and parallelism specialization for energy-efficient computing. *IEEE Micro'12*.

[12] Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. Branch-aware loop mapping on cgras. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[13] Shunning Jiang, Peitian Pan, Yanghui Ou, and Christopher Batten. Pymtl3: a python framework for open-source hardware modeling, generation, simulation, and verification. *IEEE Micro*, 40(4):58–66, 2020.

[14] Andreas Kanstein, Sebastian López Suárez, and Bjorn De Sutter. Optimizing coarse-grain reconfigurable hardware utilization through multiprocessing: An h. 264/avc decoder example. In *VLSI Circuits and Systems III*, volume 6590, page 65900C. International Society for Optics and Photonics, 2007.

[15] Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[16] Xiangyu Kong, Yi Huang, Jianfeng Zhu, Xingchen Man, Yang Liu, Chunyang Feng, Pengfei Gou, Minggui Tang, Shaojun Wei, and Leibo Liu. Mapzero: Mapping for coarse-grained reconfigurable architectures with reinforcement learning and monte-carlo tree search. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[17] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices*, 53(2):461–475, 2018.

[18] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004.

[19] Zhaoying Li, Dhananjaya Wijerathne, Xianzhang Chen, Anuj Pathania, and Tulika Mitra. Chordmap: Automated mapping of streaming applications onto cgra. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[20] Zhaoying Li, Dan Wu, Dhananjaya Wijerathne, and Tulika Mitra. Lisa: Graph neural network based portable mapping on spatial accelerators. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 444–459. IEEE, 2022.

[21] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Computing Surveys (CSUR)*, 52(6):1–39, 2019.

[22] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. *IEE Proceedings-Computers and Digital Techniques*, 150(5):255, 2003.

[23] Jackson Melchert, Kathleen Feng, Caleb Donovick, Ross Daly, Ritvik Sharma, Clark Barrett, Mark A Horowitz, Pat Hanrahan, and Priyanka Raina. Apex: A framework for automated processing element design space exploration using frequent subgraph analysis. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 33–45, 2023.

[24] Katayoun Neshatpour, Wayne Burleson, Amin Khajeh, and Houman Homayoun. Enhancing power, performance, and energy efficiency in chip multiprocessors exploiting inverse thermal dependence. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(4):778–791, 2018.

[25] Hyunchul Park, Yongjun Park, and Scott Mahlke. Polymorphic pipeline array: a flexible multicore accelerator with virtualized execution for mobile multimedia applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 370–380, 2009.

[26] Aviral Shrivastava, Jared Pager, Reiley Jeyapaul, Mahdi Hamzeh, and Sarma Vrudhula. Enabling multithreading on cgras. In *2011 International Conference on Parallel Processing*, pages 255–264. IEEE, 2011.

[27] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 127(7.2), 2012.

[28] Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 830–844, New York, NY, USA, 2021. Association for Computing Machinery.

[29] Cheng Tan, Nicolas Bohm Agostini, Tong Geng, Chenhao Xie, Jiajia Li, Ang Li, Kevin Barker, and Antonino Tumeo. Drips: Dynamic rebalancing of pipelined streaming applications on cgras. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022.

[30] Cheng Tan, Tong Geng, Chenhao Xie, Nicolas Bohm Agostini, Jiajia Li, Ang Li, Kevin Barker, and Antonino Tumeo. Dynpac: Coarse-grained, dynamic, and partially reconfigurable array for streaming applications. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 33–40. IEEE, 2021.

[31] Cheng Tan, Deepak Patil, Antonino Tumeo, Gabriel Weisz, Steve Reinhardt, and Jeff Zhang. Vecpac: A vectorizable and precision-aware cgra. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[32] Cheng Tan, Chenhao Xie, Tong Geng, Andres Marquez, Antonino Tumeo, Kevin Barker, and Ang Li. Arena: Asynchronous reconfigurable accelerator ring to enable data-centric parallel computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(12):2880–2892, 2021.

[33] Cheng Tan, Chenhao Xie, Ang Li, Kevin J Barker, and Antonino Tumeo. Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 381–388. IEEE, 2020.

[34] Shyamkumar Thoziyoor, N Muralimanohar, J Ahn, and N Jouppi. Cacti 6.5. *hpl. hp. com*, 2009.

[35] Christopher Torng, Peitian Pan, Yanghui Ou, Cheng Tan, and Christopher Batten. Ultra-elastic cgras for irregular loop specialization. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 412–425. IEEE, 2021.

[36] Dani Voitsechov and Yoav Etsion. Single-graph multiple flows: Energy efficient design alternative for gpgpus. *ACM SIGARCH computer architecture news*, 42(3):205–216, 2014.

[37] Bo Wang, Manupa Karunarathne, Aditi Kulkarni, Tulika Mitra, and Li-Shiuan Peh. Hycube: A 0.9 v 26.4 mops/mw, 290 pj/op, power efficient accelerator for iot applications. In *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 133–136. IEEE, 2019.

[38] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. Dsagen: Synthesizing programmable spatial accelerators. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 268–281. IEEE, 2020.

[39] Kehuai Wu, Andreas Kanstein, Jan Madsen, and Mladen Berekovic. Mt-adres: Multithreading on coarse-grained reconfigurable architecture. In *International Workshop on Applied Reconfigurable Computing*, pages 26–38. Springer, 2007.

[40] Tomofumi Yuki. Understanding polybench/c 3.2 kernels. In *International workshop on polyhedral compilation techniques (IMPACT)*, pages 1–5, 2014.