Appears in the Proceedings of the 27th IEEE Int'l Symp. on High-Performance Computer Architecture (HPCA-27), Feb 2021

Ultra-Elastic CGRAs for Irregular Loop Specialization

Christopher Torng[2*], Peitian Pan[1], Yanghui Ou[1], Cheng Tan[1], and Christopher Batten[1]
[1]Cornell University, Ithaca, NY [2]Stanford University, Stanford, CA
{ clt67, pp482, yo96, ct535, cbatten }@cornell.edu

***Abstract*—Reconfigurable accelerator fabrics, including coarse-grain reconfigurable arrays (CGRAs), have experienced a resurgence in interest because they allow fast-paced software algorithm development to continue evolving post-fabrication. CGRAs traditionally target regular workloads with data-level parallelism (e.g., neural networks, image processing), but once integrated into an SoC they remain idle and unused for irregular workloads. An emerging trend towards repurposing these idle resources raises important questions for how to efficiently map and execute general-purpose loops which may have irregular memory accesses, irregular control flow, and inter-iteration loop dependencies. Recent work has increasingly leveraged *elasticity* in CGRAs to mitigate the first two challenges, but elasticity alone does not address inter-iteration loop dependencies which can easily bottleneck overall performance. In this paper, we address all three challenges for irregular loop specialization and propose *ultra-elastic CGRAs (UE-CGRAs)*, a novel elastic CGRA that accelerates true-dependency bottlenecks and saves energy in irregular loops by overcoming traditional VLSI challenges. UE-CGRAs allow configurable fine-grain dynamic voltage and frequency scaling (DVFS) for each of potentially hundreds of tiny processing elements (PEs) in the CGRA, enabling chains of connected PEs to "rest" at lower voltages and frequencies to save energy, while other chains of connected PEs can "sprint" at higher voltages and frequencies to accelerate through true-dependency bottlenecks. UE-CGRAs rely on a novel ratiochronous clocking scheme carefully overlaid on the inter-PE elastic interconnect to enable low-latency crossings while *remaining fully verifiable with commercial static timing analysis tools*. We present the UE-CGRA analytical model, compiler, architectural template, and VLSI circuitry, and we demonstrate how UE-CGRAs can specialize for irregular loops and improve performance (1.42–1.50×) or energy efficiency (1.24–2.32×) with reasonable area overhead compared to traditional inelastic and elastic CGRAs, while also improving performance (1.35–3.38×) or energy efficiency (up to 1.53×) compared to a RISC-V core.**

I. Introduction

Fast-evolving application domains such as machine learning, augmented and virtual reality, and intelligence on the edge have increased the demand for energy-efficient hardware accelerators that remain flexible after fabrication. In particular, coarse-grain reconfigurable arrays (CGRAs) map dataflows to a spatial array of simple processing elements (PEs) and send data directly between PEs to reduce expensive data-movement energy in the memory hierarchy. CGRAs are well-known for efficiently targeting kernels with regular data-level parallelism in domains such as neural networks and image processing [1, 13, 15, 47, 59, 61]. However, once integrated into an SoC, they remain idle and unused for irregular workloads. There is an emerging trend towards repurposing these idle CGRA resources in general-purpose systems for accelerating both regular *and* irregular loops [18, 19, 45, 46]. Architects face three challenges for efficient acceleration of irregular loops defined by: (1) irregular memory accesses with variable latencies and non-uniform access patterns, (2) irregular control flow, and (3) performance bottlenecks due to inter-iteration loop dependencies.

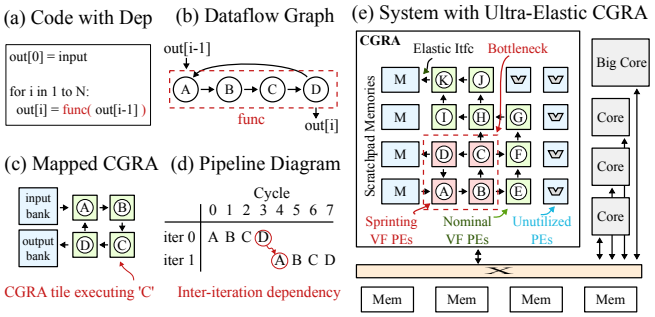


Figure 1. Irregular Loops with Inter-Iteration Loop Dependencies – CGRAs targeting irregular loops may need to address inter-iteration loop dependencies, which introduce cycles in the dataflow graph and greatly reduce throughput. (a) Toy code with a multiple-cycle inter-iteration dependency; (b) Corresponding dataflow graph with `func()` outlined in red; (c) Dataflow graph mapped to four CGRA PEs and two memory banks; (d) Pipeline diagram illustrating the inter-iteration dependency; (e) System-level view of a multicore system with a UE-CGRA coupled to the memory bus and sprinting a bottleneck region on an arbitrary kernel.

Recent work has increasingly leveraged *elasticity* in CGRAs to robustly address the first two challenges with latency-insensitive handshaking in both the memory interfaces and in the interconnect [13, 18, 19, 21, 45]. Traditional latency-sensitive CGRAs statically schedule computation at compile time and function incorrectly for any irregularity in memory access latency and/or control flow. In contrast, elastic CGRAs determine control and data flow dynamically at runtime, triggering computation when all operands have become available. Unfortunately, elastic CGRAs still struggle to achieve high performance in the presence of inter-iteration loop dependencies. Figure 1(a-d) shows a code example with a multi-cycle inter-iteration loop dependency. The corresponding dataflow graph and mapped CGRA are shown together with a pipeline diagram illustrating how throughput is limited to one iteration every four cycles. Note that this simple example could be the performance-limiting loop in a kernel with pointer-chasing behavior. While performance could still be improved by parallelizing over an outer loop (not shown) with additional resources, there is little room to mitigate the true-dependency bottleneck in the inner loop.

*This work was performed while Christopher Torng was affiliated with Cornell University.

Fine-grain dynamic voltage and frequency scaling (DVFS) has the potential to accelerate true-dependency bottlenecks in irregular loops by enabling per-PE voltage and frequency domains and allowing chains of connected PEs to "rest" at lower voltages and frequencies to save energy, while other PEs can "sprint" at higher voltages and frequencies for performance. As a simple example, the chain of true dependencies in Figure 1(d) could potentially sprint (given power slack) and complete every three cycles instead of four with the same architectural resources. Fine-grain DVFS has already been demonstrated to improve both performance and energy for multicores in academia [17, 33, 35, 56, 57] and in industry [28, 29, 38]. However, enabling fine-grain DVFS at the PE level is very challenging. Per-domain fully integrated voltage regulators and PLLs would occupy more area than the PE, and common alternative clocking schemes based on ring oscillators suffer from high phase noise. Furthermore, the resulting asynchronous crossings add synchronization latency and require specialized expertise, methodologies, and verification tools [10]. The recent exponential rise in non-recurring engineering costs motivates a greater focus on verifiability challenges [32]. Previous work has explored functional-unit-level fine-grain DVFS in various contexts, including within an out-of-order processor [51], for arbitrary logic partitionings [43], and for small circuits in isolation [62]. Some isolated works have even explored inelastic CGRAs [23, 24] — however, these works make many assumptions (e.g., per-domain PLLs, asynchronous crossings, ignoring synchronization latency) that either make the approach infeasible or significantly reduce performance and energy efficiency.

In this paper, we tackle these problems and propose *ultra-elastic CGRAs (UE-CGRAs)*, a novel elastic CGRA that accelerates true-dependency bottlenecks and saves energy in irregular loops by overcoming traditional VLSI challenges. UE-CGRAs support configurable per-PE fine-grain DVFS. Figure 1(e) shows the system-level view for the UE-CGRA platform, illustrating the fabric coupled to the system memory bus and sprinting through a true-dependency bottleneck on an arbitrary kernel. All PEs are connected with elastic buffers (i.e., PEs wait for all input operands before firing). Unutilized PEs are power-gated as in previous literature [61]. Remaining PEs are configured for different voltages and frequencies to execute the dataflow graph more efficiently.

The UE-CGRA platform relies on a *ratiochronous clocking scheme* based on [50] that is carefully overlaid on the inter-PE elastic interconnect and includes a novel elasticity-aware suppressor. This approach is not only low latency but also quantizes the frequency space and avoids asynchronous crossings, allowing for full verifiability with commercial STA tools. Our results show UE-CGRAs can improve performance (1.42–1.50×) or energy efficiency (1.24–2.32×) with reasonable area overheads compared to traditional inelastic and elastic CGRAs, while also improving performance (1.35–3.38×) or energy efficiency (up to 1.53×) over a RISC-V in-order core.

Our work challenges the conventional wisdom that kernels with irregular control flow, irregular memory accesses, and inter-iteration loop dependencies do not map efficiently to CGRAs. We also challenge the wisdom that enabling fine-grain DVFS within tiny CGRA PEs incurs too much overhead to be useful. We make the following contributions: (1) the ultra-elastic CGRA compiler, architecture, and VLSI techniques that enable a CGRA to efficiently accelerate irregular loops with inter-iteration loop dependencies;[1] (2) an analytical model for rapid UE-CGRA design space exploration; (3) the UE-CGRA compiler with a heuristic three-phase power-mapping pass; (4) a detailed architectural study of UE-CGRA design space parameters; and (5) to our knowledge, the first CGRA with per-PE fine-grain DVFS which can be *fully verified using standard static timing analysis*.

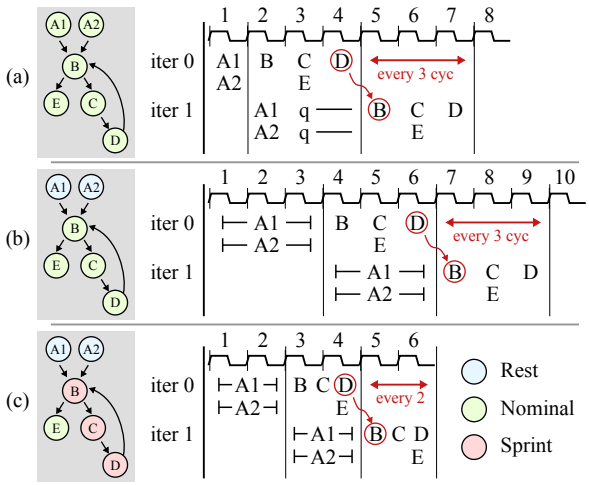


Figure 2. UE-CGRA Discrete-Event Performance Model – A toy dataflow graph with a three-node cycle provides intuition on UE-CGRA execution. (a) Execution with elastic flow control results in correct functionality (tokens wait in queue "q") with a throughput of one every three cycles; (b) Resting A1/A2 at 1/3 frequency saves energy and does not hurt throughput. (c) Resting A1/A2 to 1/2 and sprinting B/C/D by 1.5× boosts throughput to one every two cycles. Exact voltages/frequencies are used in the model but not shown here.

II. UE-CGRA Analytical Modeling

We provide intuition for the ultra-elastic CGRA computational model using first-order analytical performance and power modeling before exploring more detailed modeling in later sections. The analytical model includes discrete-event performance simulation of dataflow on the UE-CGRA computational model as well as a first-order power model.

A. Discrete-Event Performance Model

We designed a simple discrete-event simulator that models the performance of a dataflow graph (DFG) executing on both an elastic CGRA and an ultra-elastic CGRA.

Figure 2(a) illustrates a toy dataflow graph with six nodes, with three connected in a cycle. Each node in the DFG "fires" when all input tokens have arrived along the incoming edges. If an input token has not yet arrived, the node applies back-pressure and stalls. The node also stalls if a downstream node

[1] The UE-CGRA models have been released online at `https://github.com/cornell-brg/torng-uecgra-scripts-hpca2021`

is not ready to accept a new token. As time advances, tokens flow along the DFG until they exit the graph. We model two-entry queues for each edge, and we also model wire delays (i.e., tokens may only propagate after a cycle of delay). The pipeline diagram illustrates how a single inter-iteration dependency can slow the throughput to one every three cycles.

The discrete-event simulator models variation in performance by ticking nodes running at higher voltages at higher frequencies and ticking nodes running at lower voltages at lower frequencies (see Section VI for how we selected voltage-frequency pairs). For example, Figure 2(b) shows nodes A1 and A2 resting to a lower voltage corresponding to one-third the nominal frequency without impacting the kernel throughput. Finally, Figure 2(c) shows how the power slack (see energy model in Section II-B) can be used to sprint the critical DFG cycle to boost throughput to one every two cycles, reflecting a 1.5× factor in performance. Note that the A1 and A2 nodes can represent memory load PEs with SRAMs, implying that we can rest these memory banks to save energy while still executing the loads at a slow (but fast enough) rate.

Our performance model is able to execute more complex DFGs with multiple live-ins/outs (i.e., inputs/outputs to DFG), parallel fork-joins, recurrence edges, and with many nodes. Note we only model CGRAs with single-cycle operations and many details that impact mappability (e.g., routing) are abstracted away. Compared to a detailed hardware implementation, our simulator assumes that all nodes map to a unique PE and any PE can communicate with any other PE.

B. First-Order Energy Model

We also estimate the energy of a dataflow graph executing on both an elastic CGRA and an ultra-elastic CGRA using a first-order model that can be tuned against a particular technology and VLSI implementation (see Section VI).

Consider a CGRA with N_T PEs with N_{TA} configured active and N_{TI} configured inactive. The CGRA has N_S SRAM subbanks along the top and bottom edges with N_{SA} banks configured active and N_{SI} banks configured inactive.

We assume that frequency is a polynomial function of voltage (validated using circuit-level simulation, see Section VI). The frequency of each active PE and SRAM subbank is:

$$f_{TAi} = k_1 V_{TAi}^2 + k_2 V_{TAi} + k_3 \qquad (i = 1,2,...,N_{TA})$$
$$f_{SAj} = k_1 V_{SAj}^2 + k_2 V_{SAj} + k_3 \qquad (j = 1,2,...,N_{SA})$$

where k_1, k_2, and k_3 are fitted parameters, f_{TAi} is the frequency of PE i, V_{TAi} is the voltage of PE i, and so on.

We assume the throughput of an active PE or SRAM subbank is measured in iterations per second (*IPS*) and that the throughput of any active PE or SRAM subbank is equal to the throughput of the entire CGRA. This is valid because each token processed by the CGRA has visited each active PE and each SRAM subbank exactly once. Tokens that flow along cyclic recurrence edges do not count twice, as they represent inputs for the next iteration of computation:

$$IPS_{Ai} = IPS_{CGRA} \qquad (i = 1,2,...,N_{TA})$$

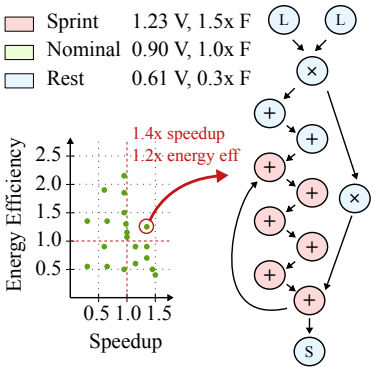


Figure 3. UE-CGRA Analytical Model Case Study – We conduct a simple sweep over individual voltages and frequency settings across all nodes, resulting in different possible configurations. This arbitrary synthetic DFG has thirteen nodes, two live-ins, one live-out, and one cycle. The circled configuration is annotated on the DFG. L = load; S = store.

Power estimation is intimately tied to both throughput and latency, and these parameters depend on the dataflow graph. We use our discrete-event performance simulator to estimate both throughput and latency. We later use this throughput estimate to calculate dynamic energy, and we also use the latency estimate to calculate static energy. Note that these raw numbers are never used in isolation. All analytical results are normalized and reported relative to another design point.

The PE and SRAM subbank powers includes both dynamic and static power and are modeled as:

$$P_{TAi} = \alpha_{i,op}\, IPS_{CGRA}\, f_{TAi}\, V_{TAi}^2 + V_{TAi}\, I_{T,leak} \qquad (i = 1,2,...,N_{TA})$$
$$P_{SAj} = \alpha_{sram}\, IPS_{CGRA}\, f_{SAj}\, V_{SAj}^2 + V_{SAj}\, I_{S,leak} \qquad (j = 1,2,...,N_{SA})$$

The factor $\alpha_{i,op}$ is in the set $\{\alpha_{mul}, \alpha_{add}, \alpha_{sll}, \alpha_{srl}, \alpha_{and}, \alpha_{cp0}\}$, where α_{op} is the relative energy of a PE executing op at nominal voltage V_N and frequency F_N compared to a PE executing `mul` at the same voltage and frequency (also validated in Section VI). Note that α_{sram} is similarly defined relative to α_{mul}.

We calculate the leakage current by assuming an architect targets leakage power to consume a certain percentage (denoted as γ) of the total power of an active PE, and that an SRAM bank's leakage current is a multiplicative factor (denoted by β) of the PE's leakage current.

$$\gamma = \frac{V_N\, I_{T,leak}}{\alpha_{mul}\, IPS_{CGRA}\, f_N\, V_N^2 + V_N\, I_{T,leak}} \qquad I_{S,leak} = \beta\, I_{T,leak}$$

We use P_{TI} and P_{SI} to refer to the power consumed by inactive PEs and SRAM banks. We assume that these are both zero, indicating that power-gated PEs consume no power.

The total power is the aggregate across PEs and SRAMs:

$$P_{total} = \sum_{i=1}^{N_{TA}} P_{TAi} + \sum_{j=1}^{N_{SA}} P_{SAj} + N_{TI}\,(P_{TI}) + N_{SI}\,(P_{SI})$$

C. Analytical Case Study

The analytical model can quickly explore the UE-CGRA design space. Figure 3 sweeps different voltage and frequency settings for each node across all nodes in the DFG. The energy for each node is modeled with its specific operator running at its specific voltage and frequency. We make the following assumptions for parameters in the UE-CGRA analytical energy model: k_1 = -1161.6, k_2 = 4056.9, k_3 = 1689.1,

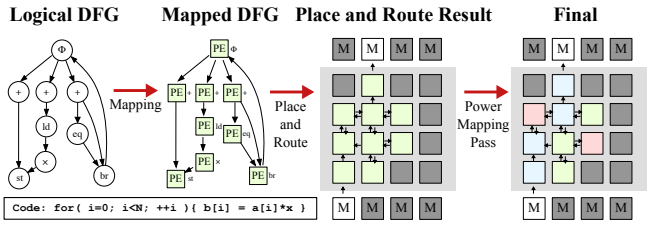


Figure 4. UE-CGRA Compiler Flow – The UE-CGRA compiler builds on LLVM and transforms a C/C++ program into a logical DFG, maps to physical PEs, places/routes onto the array, selects power modes per PE, and generates the final bitstream.

V_N = 0.9 V, V_{min} = 0.61 V, V_{max} = 1.23 V, f_N = 750 MHz, γ = 0.1, β = 2.0, α_{sram} = 0.82 (per 4 kB subbank), α_{mul} = 1.0, α_{add} = 0.30, α_{sll} = 0.37, α_{srl} = 0.35, α_{cp0} = 0.23, α_{and} = 0.30, α_{or} = 0.33, α_{xor} = 0.42, α_{eq} = 0.23, α_{ne} = 0.23, α_{gt} = 0.25, α_{geq} = 0.25, α_{lt} = 0.25, α_{leq} = 0.25, α_{bps} = 0.11. These parameters are derived from VLSI modeling for the target voltage range and system described in Section VII.

The circled point combines sprinting and resting for 1.4× speedup and 1.2× energy efficiency. Note that sprinting the six-node cycle increases energy, but resting non-critical nodes reduces energy (in particular, live-ins and live-outs represent power-hungry SRAMs). The results also suggest that resting can enable 2.2× energy efficiency at similar performance.

III. UE-CGRA Compiler

The UE-CGRA compiler is responsible for transforming the C/C++ source code of a compute kernel for implementation (i.e., generating the bitstream) on the UE-CGRA architecture (see Section IV). Figure 4 overviews the compiler toolchain, which builds on LLVM to transform simple C programs into logical dataflow graphs, conducts a simple mapping of nodes to physical PEs, places and routes the design onto the UE-CGRA, and then conducts a heuristic power mapping pass to configure the voltages and frequencies to optimize performance and energy efficiency. Because the LLVM and place/route phases are similar to related work (e.g., [30]), we focus our discussion on the power mapping pass. See Section VI-A for more detail on the LLVM passes.

The compiler power-mapping pass automatically selects DVFS power modes, one mode per PE. The pass begins by power gating unused PEs as is common in related literature [61]. We then apply the three-phase mapping algorithm described in Figure 5, which leverages the UE-CGRA analytical model for energy-delay estimates and employs heuristics to iteratively search for a mapping with high performance and/or high energy efficiency. The algorithm is simple and not intended as a formal solution for the general power-mapping problem. However, it is sufficient to explore the potential of the UE-CGRA design space.

Complexity-Reduction Phase – Naively, selecting from M possible power modes (e.g., rest, nominal, sprint) for each of N logical DFG nodes has a worst-case time complexity of $O(M^N)$, which grows quickly for larger DFGs. To reduce the search space, we take advantage of the producer-consumer relationship in a DFG and note that the throughput of an entire

```
1:  procedure PowerMapping(N, T)
2:      for each node n in N do
3:          M(n) = V(sprint)
4:      G = GroupNodes(N)
5:      for each group g in G do
6:          M(g) = V(rest)
7:          if MeasureEnergyDelay(CGRA) < 1.0 then
8:              M(g) = V(nominal)
9:              if MeasureEnergyDelay(CGRA) < 1.0 then
10:                 M(g) = V(sprint)
11:     for each PE t in T do
12:         ConstrainPEModes(t)
```

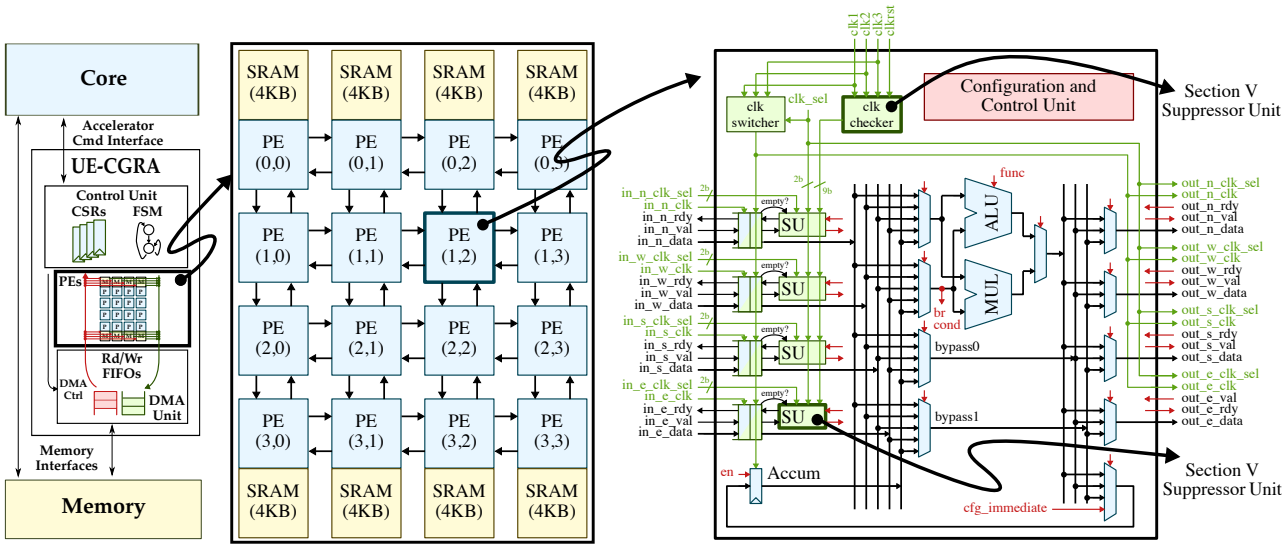
Figure 5. UE-CGRA Compiler Power-Mapping Algorithm – The PEs are mapped to rest, nominal, and sprint power modes. Heuristics are applied to maximize energy-delay product. N = set of all logical DFG nodes; T = set of all physical PEs; $M(n)$ = power mode of node n; $V(x)$ = voltage for mode x. The *MeasureEnergyDelay*(*CGRA*) result is relative to the previous best (i.e., < 1.0 is worse).

chain is determined by the slowest PE. Specifically, a singly connected chain of nodes should run at the same frequency and can be treated as living in a single logical power domain. We group all such nodes (i.e., *GroupNodes*()) to effectively reduce N. Note that nodes with multiple inputs and outputs remain ungrouped from other nodes. The next two phases of the algorithm apply heuristics for energy and delay and further reduce the worst-case time complexity to $O(NM)$.

Energy-Delay-Optimization Phase – In this phase, the compiler initializes each logical node n to sprint voltage such that power mode $M(n) = V(sprint)$. We then attempt to rest each group of nodes for an improved energy-delay product. At the start since all nodes are initialized to sprint, performance is already maximized and the compiler slowly trades performance for greater factors of energy efficiency. The algorithm greedily tries to rest first for the greatest potential energy-efficiency benefit before trying nominal, and then rolling back to sprint. The *MeasureEnergyDelay()* function estimates results using the analytical model by setting all parameters (in Section II-C) and all power modes for all nodes. This phase produces a power mapping and an energy-delay product strictly less than that of the starting configuration.

Constraint Phase – Logical nodes in the DFG may *fold* onto the same physical PE (i.e., since size(N) ≥ size(T)) and therefore be limited to run at the same voltage and frequency. For example, a PE may execute a multiply with two inputs while bypassing a third unrelated input to an adjacent PE (i.e., routing through a busy PE). In this phase, the algorithm identifies all nodes mapped to PE t and chooses a single power mode in the event of disagreement. The *ConstrainPEModes()* function implements a small energy-delay optimization search across the options, estimating each option internally with the *MeasureEnergyDelay()* function.

Summary – This algorithm prioritizes performance by seeding the initial state to the maximum performance point (i.e., all sprint) to generate a *performance-optimized mapping*. A variation of the algorithm can also prioritize energy by initializing all nodes to nominal mode to generate an *energy-optimized mapping*. This algorithm is simple but ef-



(a) UE-CGRA System (b) 4x4 UE-CGRA PE Array (c) UE-CGRA PE

Figure 6. UE-CGRA Architecture – Detailed block diagrams: (a) a system that offloads computation to a UE-CGRA; CSR: control status registers. (b) a 4×4 UE-CGRA PE array with 4 kB SRAM banks; (c) a UE-CGRA PE contains input queues, muxing, an operator block, a configuration register, a multi-purpose register (for accumulation and storing constants), a control unit with support for *merge* functionality *phi* and branches *br*. Supported operations: cp0, cp1, add, sub, sll, srl, and, or, xor, eq, ne, gt, geq, lt, leq, mul, phi, br, nop.

fective. We see our algorithm as just a heuristic starting point that demonstrates promise for UE-CGRAs. More sophisticated variations that map iteratively with physical constraints would be more effective and are interesting future work.

IV. UE-CGRA Architecture

In this section, we describe the UE-CGRA architectural template which enables repurposing the CGRA to accelerate both regular *and* irregular workloads. We conduct various sweeps to draw insights about the architectural design space.

A. Architectural Template

Figure 6 illustrates the block diagram for a system containing a 4×4 UE-CGRA. To offload computation, the processor writes the control status registers (CSRs) in the CGRA through the accelerator command interface and sets up the base addresses and sizes for the DMA unit to fetch the configuration bitstream and the data. Data is loaded into the SRAM banks in the PE array. Another CSR is then written to initiate computation. The complete UE-CGRA is composed of a control unit, a DMA unit with read and write queues, and an array of UE-CGRA PEs interconnected with queues. The PEs along the north and south perimeter contain SRAM banks and are the only PEs capable of memory operations. The PE is carefully architected to enable both compute and bypassing of data (i.e., routing) in the same cycle. The configuration phase leverages the existing data network to forward configuration messages systolically through the array from top to bottom. There are 26 configuration bits, which fits in an inter-PE message. Each block is described in more detail below.

Input Queues and Registers – The input queues from each cardinal direction are elastic and have two entries to avoid creating unnecessary pipeline bubbles [41] when all PEs are communicating on the same synchronous clock. They are also designed to correctly interface clock domains with known phase relationships (see Section V). A single-entry multi-purpose register is included to store configured constants, accumulate values, and implement phi node behavior for recurrence edges (i.e., cycles in the DFG).

Muxing and Operators – Four five-input muxes select between the four input queues and the multi-purpose register. These supply the operands for the compute operator as well as for two bypass paths. Bypassing enables any input queue to forward data to any output, allowing messages to route through PEs executing other operations. Five three-input muxes select between the compute operator output and the bypass messages before forwarding the message towards one of the four cardinal directions or towards the multi-purpose register. The compute operator supports the following operations in a 32-bit datapath: cp0, cp1, add, sub, sll, srl, and, or, xor, eq, ne, gt, geq, lt, leq, mul, phi, br, nop. These operations include control flow (handled as data flow). The multiply operation truncates the upper half of the result so that the inputs and outputs have identical bitwidths.

Merge and Branch Support – Phi nodes are *merge* operations that fire when either of two input messages arrive. Merge behavior is paired with control unit logic that sends a single valid handshake after reset. This enables PEs to initialize DFG cycles (i.e., iteration zero) with valid data. Because PEs wait for inputs, if none of these inputs were initialized, the PE would never fire. Branching is decoupled from computation of the branch condition. A PE configured to branch accepts two handshakes for the data and select, which determines which of two outputs to forward data to. The branch bit is tapped from one operand which is then used as the condition. Note that we convert all control flow into data flow.

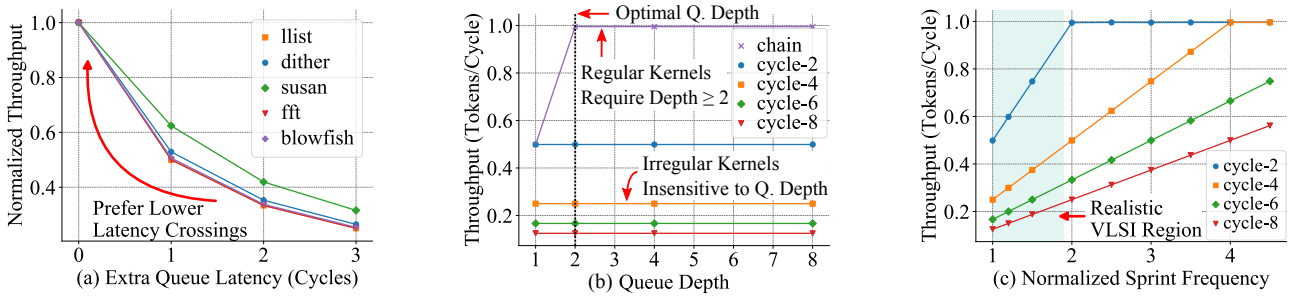


Figure 7. Performance Sweeps – Methodology for all three figures (Section VI) based on RTL simulation using performance-optimized power mapping for both kernels and synthetic microbenchmarks (see raw DFGs in Figure 14). (a) Throughput with varying inter-PE latency (cycles per hop). (b) Throughput with varying queue depth. `cycle-N` = a synthetic irregular microbenchmark with *N* nodes connected in a cycle. `chain` = a synthetic regular microbenchmark with no cycles. (c) Throughput with varying sprinting frequencies. In the highlighted realistic VLSI region, faster sprint provides linear returns.

B. Q&A: Impact of inter-PE latency on performance?

An architect's first instinct may be to use asynchronous FIFOs to cross between clock domains, but these add two to three cycles of inter-PE synchronization latency [10] with an unclear impact on performance.

Dataflow architectures communicate at fine granularity (e.g., every cycle) and are very sensitive to inter-PE latency. Figure 7(a) quantifies this impact and shows how a two-cycle synchronization latency degrades throughput by a factor of three for DFGs with inter-iteration dependencies. A single token must propagate completely around the DFG cycle before the next iteration can begin. As hops slow down, the throughput of the critical DFG cycle (and therefore the entire kernel) also slows down. As a result, **performant dataflow architectures must reduce inter-PE synchronization latency as much as possible**. Note that asynchronous FIFOs only pay a penalty when empty or full. Unfortunately, a token flowing around a DFG cycle will repeatedly encounter PEs with empty queues (i.e., the worst case), meaning that **inter-PE asynchronous FIFOs will reduce throughput by a factor of 3–4**×. High performance requires zero inter-PE latency.

C. Q&A: Impact of queue depth on performance?

Elastic architectures typically use two-element queues to avoid creating unnecessary pipeline bubbles [41]. However, since UE-CGRA PEs may fill their downstream queues more slowly (or quickly) than in an E-CGRA, the question of queue depth is important to guarantee maximum throughput.

A UE-CGRA is meant to augment an existing CGRA to accelerate both regular *and* irregular workloads. Therefore, at least two-element queues are required to support full throughput for regular workloads. For irregular workloads, we study queue depth in a UE-CGRA running a synthetic irregular microbenchmark (Figure 7(b)) and show that **no amount of deeper queuing affects the throughput of irregular kernels with inter-iteration dependencies**. While surprising at first, this is in fact intuitive. Specifically, throughput is determined by the latency of a single token propagating around the longest DFG cycle (i.e., with the most nodes) before the next iteration can fire. The queues in these PEs are always empty and so depth has no impact (in fact, even a single element is sufficient). Therefore, we find that **two-element queues are reasonable to support both regular and irregular kernels**.

D. Q&A: Which frequencies should be supported?

While an architect may desire many frequency levels for greater flexibility, the VLSI overhead for each additional level can grow quickly (e.g., clock generation, global clock networks, power grids, more complex static timing analysis). This subsection studies which levels should be prioritized over others given a conservative limit of only three levels.

The nominal level is required (e.g., 0.9 V in TSMC 28). Recent literature (in non-CGRA settings) suggests that having both a rest and sprint voltage will enable both saving energy and improving performance [17, 33, 42, 57]. However, we cannot sprint too high (e.g., to avoid transistor breakdown) nor rest too low (e.g., to avoid near-threshold). In TSMC 28, from a reasonable range for resting (0.6–0.9 V) and sprinting (0.9–1.3 V), we must choose two levels.

For the sprint level, Figure 7(c) sweeps sprint frequencies for a UE-CGRA running synthetic irregular microbenchmarks (same as in Figure 7(b)) and a performance-optimized compiler mapping. Speedup is proportional to frequency (i.e., the token propagates around the ring more quickly) until performance hits a throughput ceiling (i.e., the rate of incoming producer data is set to one token per nominal cycle here). DFGs with longer cycles require higher frequencies to hit the ceiling. However, only a limited range of frequencies is realistic from a VLSI perspective (e.g., sprinting to a high 1.3 V in TSMC 28 only increases current drive for roughly a 1.58× frequency boost, see SPICE modeling in Section VI-B). In a realistic VLSI setting, **higher sprint frequency linearly improves throughput** by speeding up the critical DFG cycle.

For the rest level, our SPICE modeling suggests that the lowest rest voltage (i.e., 0.6 V) would decrease current drive for roughly a 3.0× slower frequency (and a 7× power reduction). While this may seem slow, the opportunity to rest at this level can be surprisingly high because any DFG with an inter-iteration dependency of three or more ops could rest non-critical PEs and SRAM banks (e.g., see Figure 2). Therefore, it is likely that **the opportunity to rest aggressively in a realistic VLSI setting is high** without impacting throughput.

E. Summary

The UE-CGRA architecture should prioritize low-latency inter-PE crossings (ideally zero), queue depth should be two elements, higher sprint modes can be chosen for linear performance returns, and the lowest rest mode should be chosen, given the high likelihood for resting in irregular kernels. A reasonable set of voltages in TSMC 28 might therefore be (0.60 V, 0.90 V, 1.30 V), before further VLSI considerations.

V. UE-CGRA VLSI

The previous sections assumed UE-CGRA platform VLSI support. In this section, we describe the UE-CGRA VLSI circuitry to enable per-PE fine-grain DVFS with reasonable overheads. Figures 8 illustrates the primary components of our clocking scheme and physical design approach.

Ratiochronous Clock-Domain Crossings – UE-CGRA PEs communicate synchronously over ratiochronous clock-domain crossings. The ratiochronous design pattern enforces *rational* clocking relationships across domains (e.g., frequency ratios of 1-to-3, 2-to-3). This requirement is less flexible than a fully asynchronous approach (i.e., ratiochronous clocking quantizes an otherwise infinite space of possible clock relationships). The ratiochronous family of cross-domain interfaces was initially proposed by Sarmenta et al. who published the seminal paper on rational clocking [50], and it has since been explored primarily in the asynchronous community for small circuits (see Section IX). Enforcing ratiochronous crossings has minor impact on an architect's choice of frequencies. In this paper, we adjust our preferred selection of voltages from (0.60 V, 0.90 V, 1.30 V) to (0.61 V, 0.90 V, 1.23 V) to convert our clock ratio from "1.9-to-3-to-8.5" to "2-to-3-to-9". This reduces sprint frequency by 5% according to our SPICE models in Section VI-B.

Suppressor Unit – Ratiochronous crossings have phasic relationships that may require suppression of "unsafe" edges as shown in Figure 8(a). For example, clock edges at a 2-to-3 crossing repeat periodically at the least common multiple of six. Several edges are not aligned and are "unsafe" for transmitting data. Figure 8(c) implements a traditional unsafe-edge detector [50] for each crossing between three clock domains. Because CGRA performance is very sensitive to frequent and periodic stalling, we propose a *novel elasticity-aware suppressor unit* that enables safe crossings in the presence of backpressure. Specifically, Figure 8(d) shows how the empty signal from the input queues is used to implement two edge detectors that *allow handshakes on unsafe edges* as long as data has been enqueued for longer than one local clock cycle, which can prevent unnecessary stalling.

Bisynchronous Queues – We select a simple two-element bisynchronous normal queue to interface between two PEs in different clock domains. Messages sent are source-synchronous, meaning the write clock is sent with the data. We specifically do *not* use asynchronous queues as is assumed in most literature [24, 51, 62], as these complicate verification [10] and add two-to-three-cycle synchronization latency penalties which significantly reduce performance.

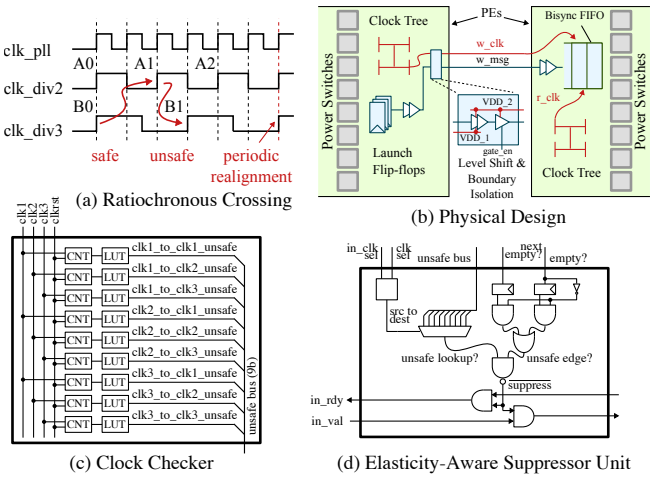


Figure 8. Ratiochronous Crossings and Power Domains – (a) Rational clocks with 2-to-3 ratio and edges uniquely numbered (before repeating periodically). The first crossing is *safe* since the propagation time from edge B0 to edge A1 is a full (receiver) clock cycle from edge A0 to edge A1; the second crossing is *unsafe* and too aggressive to meet timing. (b) Two PEs with level shifters and boundary isolation. Source-synchronous messages travel to a neighboring PE through level shifters/isolation and write the bisync queue, where the consumer reads it (entire path checked by STA). (c) Clock checker to detect safe/unsafe edges in ratiochronous clock-domain crossings. (d) Suppressor unit to disable handshakes on unsafe cycles.

Static Timing Analysis (STA) – Because ratiochronous design quantizes the frequency space, UE-CGRAs are fully verifiable with commercial STA tools by checking the cross-product of each domain at each frequency. Our UE-CGRA template further reduces the complexity of the verification space by leveraging suppressors to eliminate unsafe crossings at the architecture level. STA tools need only check safe crossings, which significantly simplifies timing constraints.

Clock Dividers and Switchers – We generate rational clocks with standard 50%-duty clock dividers (e.g., divide by two, divide by three) [48] and distribute to all PEs. Each PE then selects between clocks with a traditional glitchless clock switcher. At the system level, we then align all clocks with a two-phase reset: a dedicated clock reset (i.e., Figure 8(c) "clkrst") aligns all clock dividers and switchers, which generates the edges to synchronously reset all state within PEs.

Hierarchical Clock-Network Gating – Three clock networks in the UE-CGRA distribute the divided clocks (i.e., rest, nominal, and sprint) to each PE. *Gating these clocks is critical* to the energy efficiency of the UE-CGRA. We first power gate inactive PEs to eliminate their local clock power as in the literature [61]. We then adopt a hierarchical clock-network gating approach to reduce toggles in large unselected portions of the global clock networks, similar to [49]. Specifically we cluster PEs (e.g., 4×4 clusters) and gate each cluster's portion of the global clock network with a 1-bit configuration register. Because the compiler is statically aware of all PE clocks, it can gate with perfect knowledge (e.g., it can gate the entire sprinting clock if no PEs are sprinting).

```
while(hd) {
  if(hd->d==tgt)
    return hd->d;
  else
    hd=hd->nxt;
}
return -1;
```
(a) llist

```
for(i=0;i<N;++i){
  out = src[i]+err;
  if(out>127) {
    pixel = 0xFF;
    err = out-pixel;
  } else {
    pixel = 0;
    err = out;
  }
  dest[i] = pixel;
}
```
(b) dither

```
for(x=-S;x<=N;x++){
  bright=total+
   *ip++;
  tmp=*dpt+++
   *(cp-bright);
  area+=tmp;
  total+=tmp*bright;
}
```
(c) susan

```
for(k=0;k<G;++k){
  t_r=Wr*r[2*j*G+G+k]
   -Wi*i[2*j*G+G+k];
  t_i=Wi*r[2*j*G+G+k]
   +Wr*i[2*j*G+G+k];
  r[2*j*G+G+k]=
   r[2*j*G+k]-t_r;
  r[2*j*G+k]+=t_r;
  i[2*j*G+G+k]=
   i[2*j*G+k]-t_i;
  i[2*j*G+k]+=t_i;
}
```
(d) fft

```
for (i=0;i<21;++i){
  BF_ENC(right,left,
   s,p[i]);
  temp=right;
  right=left;
  left=temp;
}
```
(e) blowfish

Figure 9. Target Kernel Loops – Code for each inner loop and their inter-iteration dependencies (e.g., llist `hd`, blowfish `left/right`).

Multi-Rail Supply Voltages – The research community has studied fine-grain DVFS with multi-rail supply voltages [11, 42, 57] and fully integrated voltage regulators [14, 16, 17, 25, 33, 35, 54, 56]. In this work, we select a traditional multi-rail supply scheme with three voltages. This increases pin count and hierarchical power grid overhead but enables fast reconfiguration on the order of nanoseconds [57] (also verified in-house in TSMC28 for a PE at power signoff).

Voltage Level-Shifting Considerations – Driving signals directly between voltage domains can be functionally incorrect (e.g., 0.6 V driving 1.2 V). Modern commercial ASIC library vendors provide universal up-down level-shifters (e.g., 100s of mV in one FO4 delay). However, driving from near-threshold voltage can increase this latency to 1–10 ns even with cutting-edge circuit design [34]. UE-CGRAs are not designed for near- or sub-threshold voltage domains and rely only on commercially available level shifters operating above threshold. Figure 8(b) shows how these cells level-shift outbound signals from the PE, directly behind the boundary-isolation cells (with gate_en) which are already present for fine-grain power-gating in modern CGRAs [61]. This ensures that outbound signals are always driven to known values.

VI. Methodology

We use a vertically integrated research methodology spanning software, architecture, and VLSI and describe our benchmarks, compiler, architectural modeling, and VLSI.

A. Benchmarks and Compiler

CGRA compilers typically identify small 10-instruction regions that can be reused 10K+ times [18, 19]). Figure 9 shows the inner loop code for the five benchmarks used in our evaluation and their inter-iteration dependencies. We map only the innermost loops. We choose benchmarks small enough to fit in our 8×8 UE-CGRA while remaining suitable for detailed evaluation. *llist* is a linked-list search kernel. *dither* runs grayscale Floyd-Steinberg image dithering. The remaining few kernels are derived from previous literature on elastic CGRAs [21], including *susan*, an automotive image recognition smoothing kernel, and *bf*, a block-cipher security kernel.

Our compiler is implemented as an LLVM pass (v3.8.0). We generate a DFG for the innermost loop of each kernel and structurally map the DFG onto the CGRA. We implement a control dataflow graph (CDFG) analysis pass to generate the CDFG (i.e., with both control and data dependency edges) for each kernel. The CDFG is tuned (e.g., control-dependency edges are converted to data-dependency edges with phi and branching support) to generate a new DFG containing only the set of operations supported by the UE-CGRA. We then place one DFG node at a time onto PEs in the UE-CGRA. For each node, a valid path to route dependencies is calculated with Dijkstra's algorithm.

B. Architecture and VLSI Modeling

We designed a parameterizable elastic CGRA (E-CGRA) and UE-CGRA RTL generator within a Python-based hardware modeling framework [26, 27, 37]. The E-CGRA is similar to [21]. We translated to Verilog and pushed each 8×8 CGRA through an ASIC toolflow at 750 MHz using Synopsys and Cadence tools (i.e., Design Compiler, Innovus, PrimeTime PX, and VCS) in TSMC 28 nm. We ran synthesis, place-and-route, gate-level simulation, and power estimation.

We also ran SPICE-level simulations to help determine the relationship between voltage and frequency for our process technology across different operating modes. We used 21 delay stages consisting of multiple FO4-loaded inverters, NAND, and NOR gates connected in a loop, such that the total delay in the loop matches our gate-level cycle time for a given voltage. We used the change in delay vs. supply voltage as a model for PE voltage-frequency scaling.

C. Energy Modeling

We model energy for our 8×8 E-CGRA and our 8×8 UE-CGRA with vectored SDF-annotated gate-level power estimation using Synopsys VCS and PrimeTime PX at nominal voltage, followed by first-order power-scaling equations for different voltages. We first simulated each benchmark separately on both gate-level models at nominal voltage. We then used both sets of energy breakdowns to model the intended UE-CGRA with rest and sprint voltages. Specifically, the total UE-CGRA energy is the sum of energies in all E-CGRA PEs, the suppression logic, and the clock switcher in all 64 UE-CGRA PEs. We scaled each PE to the new voltage to first order, re-accounted for leakage, and added global clock energy (which is not voltage-scaled). The gate-level simulation is driven by 1000 iterations of random input data for *llist*, *fft*, *susan*, *dither* and 32 iterations for *bf*. We model hierarchical clock-network gating (see Section V) using post-PnR SDF-annotated power reports. We extract local (PE) clock power and global clock network power (rest, nominal, sprint) and also analyze the power breakdown of all clock tree buffers. Then for each CGRA configuration, we subtract (i.e., gate)

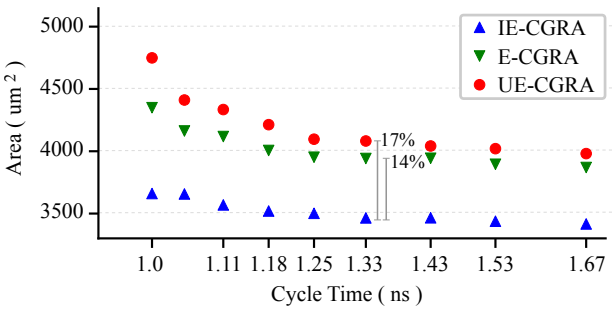


Figure 10. PE Area vs. Cycle Latency – PE areas of IE-CGRA, E-CGRA, and UE-CGRA across different cycle time targets.

portions of the global clock network based on the average power of a clock buffer multiplied by the number of toggling clock buffers in each tree (by cross-checking with the compiler to see which PEs use that clock). For example, if two-thirds of the PEs use nominal clock, then we estimate the number of clock buffers in the tree driving those PEs, and if the compiler decides that no PEs use the sprint clock then that entire network can be gated. Finally, note that our gate-level power estimates for the CGRA PE (see Section VII-A) propagate back to the UE-CGRA analytical model as α_{op} values.

D. System Integration Modeling

We model a system with an in-order RV32IM core coupled with a CGRA. The core is implemented similarly to our CGRAs and meets 750 MHz timing post-layout. We model performance for the CPU in isolation by compiling each kernel with the RISC-V toolchain and simulating in RTL, and we measure energy as discussed earlier with Synopsys PTPX. To model performance of a CPU offloading a benchmark onto a CGRA, we estimate to first order the reconfiguration and data-loading overheads based on benchmark properties. Specifically, we assume that all accesses to the L1 cache are hits and that the sustained bandwidth between the CGRA and the memory system is 128 bits/cycle (similar to [18, 19]).

VII. Results

In this section, we evaluate UE-CGRAs against inelastic CGRAs (IE-CGRA) and elastic CGRAs (E-CGRA) executing a set of irregular kernels that fit in an 8×8 CGRA for detailed simulation. We then finish with a first-order system-level analysis with a RISC-V core. Note that while these studied CGRAs have limited resources and can only fit simpler kernels, modern CGRAs are beginning to scale in size (e.g., 500+ PEs [2, 61]) to accelerate regular ML workloads. We will show that the overhead to convert to an E-CGRA or UE-CGRA is reasonable to also support irregular workloads.

A. PE Area and Energy

We first consider a single IE-CGRA PE (no elasticity, no DVFS), an E-CGRA PE, and a UE-CGRA PE in a 28 nm technology. Figure 10 shows post-PnR area for a sweep of clock targets. Area increases as expected for more aggressive clocks. In general, the E-CGRA PE and UE-CGRA PE have similar area with 14% and 17% overhead over the IE-CGRA

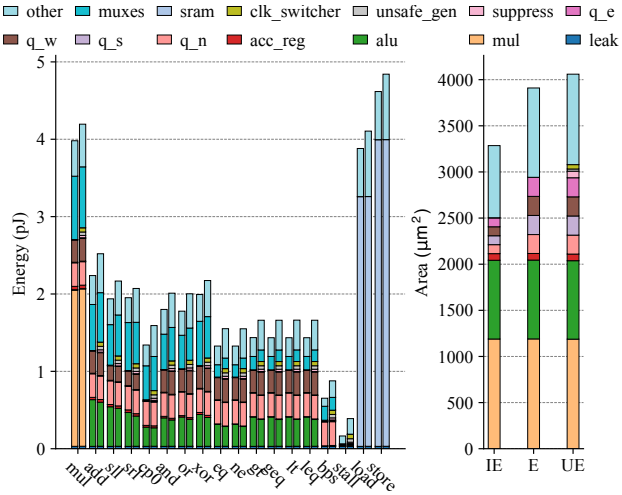


Figure 11. PE Energy and Area Breakdowns – PE energy breakdowns for each configurable operation for E-CGRA (left bar) and UE-CGRA (right bar) for a 750 MHz clock in TSMC 28 nm. PE area breakdowns are also shown for IE-CGRA, E-CGRA, and UE-CGRA. {q_e, q_w, q_s, q_n} = queues; reg = multi-purpose register; {unsafe_gen, suppress} = UE-CGRA VLSI circuitry; mul = multiply; {sll, srl} = shifts; {and, or, xor} = bitwise ops; {eq, ne, gt, geq, lt, leq}= compare; bps = bypass; {load, store} = SRAM access.

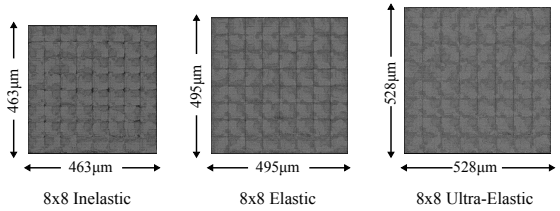


Figure 12. CGRA Layouts – CGRA layout for IE-CGRA, E-CGRA, and UE-CGRA targeting 750 MHz in TSMC 28 nm.

PE at a 750 MHz target (1.33 ns). Notably, the E-CGRA has similar overheads over the IE-CGRA as in [21].

Figure 11 breaks down area for each PE and also energy for a single E-CGRA PE and UE-CGRA PE across all operations. The area for UE-CGRA-specific logic (e.g., unsafe crossing suppression) is very small. On average, the UE-CGRA PE consumes 21% more energy across all operations compared to an E-CGRA PE. The unsafe clock-domain crossing suppression logic (field unsafe_gen and suppress in Figure 11) contributes to a minimal 1.3% energy overhead. The remaining difference is from the three clock networks entering the PE. Note that our full-CGRA results in Section VII-C will reflect hierarchical clock-network gating as described in Section V to reduce this energy for inactive PEs.

B. CGRA Area, Cycle Time, and Power Breakdown

Figure 12 shows the post-PnR layout of the three 8×8 CGRAs. The IE-CGRA has the smallest area (463×463 µm) due to its simplicity and lack of flow control. The E-CGRA has slightly larger area (495×495 µm) with flexible, hardware-managed flow control. The UE-CGRA has an area overhead of 14% compared to an E-CGRA due to three global clock networks, global clock dividers, and per-

TABLE I. POWER BREAKDOWNS

		PE Logic Power	PE Clock Power	Global Clock Sprint	Global Clock Nominal	Global Clock Rest	Total Clock Power	Total Power
	E-CGRA w/o P+H	1.66	1.70		0.24		1.94	3.60
	E-CGRA w/o H	0.94	0.80		0.24		1.04	1.98
	E-CGRA	0.94	0.80		0.10		0.91	1.85
POpt	UE-CGRA w/o P+H	2.31	1.91	0.54	0.36	0.12	2.93	5.29
POpt	UE-CGRA w/o H	1.43	1.13	0.54	0.36	0.12	2.15	3.63
POpt	UE-CGRA	1.43	1.13	0.16	0.04	<0.01	1.34	2.82
EOpt	UE-CGRA w/o P+H	1.70	1.37	0.53	0.36	0.12	2.38	4.12
EOpt	UE-CGRA w/o H	0.81	0.59	0.53	0.36	0.12	1.60	2.47
EOpt	UE-CGRA	0.81	0.59	<0.01	0.15	<0.01	0.74	1.61

Power (mW) derived from SDF-annotated post-PnR design running the dither kernel (Popt/Eopt, Figure 14 (g/h)). P: unused PEs are power gated (and therefore clock gated); H: hierarchical clock gating (see Section V). PE clock power: the intra-PE clock network power (clock buffers and DFFs in PEs). Global clock: the inter-PE clock network power (clock buffers outside PEs). POpt/EOpt: see Section VII-C.

PE clock switchers and suppression logic. All three CGRAs target a 750 MHz nominal clock frequency and meet timing.

Table I shows the post-PnR power breakdown of the *dither* kernel running on the E-CGRA and UE-CGRA with and without power gating of idle PEs and the hierarchical clock-network gating in Section V. The first row for each CGRA shows that the clock network (with no gating) accounts for about half the overall power, agreeing with the conventional wisdom that clock networks have significant overhead. Note that both UE-CGRAs have global clock power (i.e., not including the PE portion) about 4× that of the E-CGRA, and accounts for 19% of the total power of UE-CGRA POpt.

The second row power gates inactive PEs which also eliminates their *local* clock network power, significantly reducing overall power. Finally, the third row applies hierarchical clock gating as described in Sections V and VI-C to reduce *global* clock network power. Comparing to the most optimized E-CGRA (third row), successively gating the UE-CGRA (POpt) starts at 2.86× power overhead and reduces first to 1.96× and then to 1.52× with both methods of gating. Note that UE-CGRA POpt has higher power than the E-CGRA, but UE-CGRA EOpt has lower power, showing how the compiler can target different power budgets.

C. CGRA Performance and Energy

Performance and gate-level energy results are listed in Table II for the 8 × 8 UE-CGRA relative to the baseline 8 × 8 E-CGRA. These results include hierarchical clock-network gating as described in Section V. Figure 14 visualizes the energy consumption of each PE in the baseline E-CGRA and the UE-CGRAs with performance/energy-optimized mappings on kernels *llist* and *dither*. These two mappings are generated by configuring the initial state of the power-mapping algorithm to prioritize either performance or energy. We do not include IE-CGRA in this comparison because it requires a radically different kernel mapping with extra routing PEs and slack matching and is therefore not a fair comparison. Note that the kernels do not fully utilize the CGRA, and this is a problem for CGRAs in general (our average is 65% utilized).

TABLE II. UE-CGRA PERFORMANCE AND ENERGY

Mapping	Energy-Optimized Perf	Energy-Optimized E.Eff	Performance-Optimized Perf	Performance-Optimized E.Eff
llist	1.00	1.50	1.49	1.09
dither	1.00	1.24	1.42	1.00
susan	1.00	1.73	1.50	1.19
fft	1.00	2.32	1.49	2.02
bf	0.87	1.32	1.44	1.05

Performance (iterations/s) from RTL simulation and energy efficiency (iterations/J) from post-PnR power estimation with SDF-annotated GL simulations. Both relative to 8 × 8 E-CGRA.

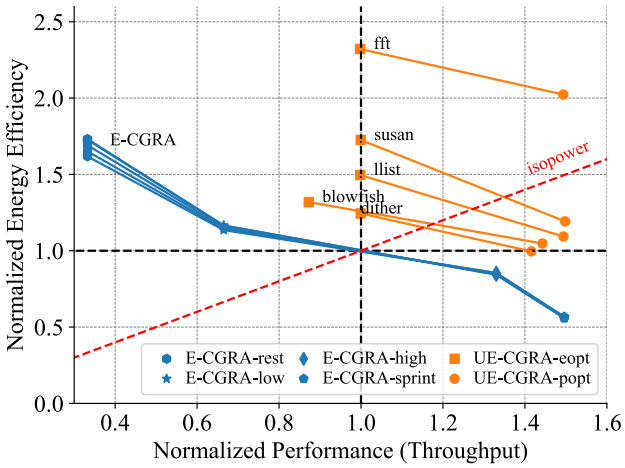


Figure 13. Normalized Energy Efficiency vs. Performance – Performance (iterations/s) and energy efficiency (iterations/J) plotted relative to an E-CGRA baseline running at nominal VF (0.90V, 1.00X freq.). Blue curves scale the entire E-CGRA (e.g., to full rest, full sprint, and in between), and orange curves indicate fine-grain DVFS with UE-CGRA. rest = all fully rested (0.61V, 0.33X freq.); low = slightly rested (0.80V, 067X freq.); high = slightly sprinted (1.0V, 1.33X freq.); sprint = all fully sprinted (1.23V, 1.50X freq.).

Energy-Optimized Mapping (EOpt) – The energy-optimized UE-CGRA prioritizes resting PEs while avoiding performance loss. Table II shows that an 8 × 8 UE-CGRA can improve energy efficiency by up to 2.32× over an 8 × 8 E-CGRA for our kernels. The energy-optimized power mapping identifies PEs on non-critical paths that trigger only once every few cycles and configures these PEs to rest at lower power modes while still (slowly) computing. The energy contours show lighter colors for this mapping in the UE-CGRA. Most kernels have acceptable performance degradation as less-critical PEs operate at lower frequencies (0.8× to 1.0×).

Performance-Optimized Mapping (POpt) – The performance-optimized UE-CGRA sprints PEs in the recurrence loop while also resting less-critical PEs for energy efficiency. Table II shows that kernels can achieve up to 1.77× speedup. From Figure 14(g), the performance-optimized mapping achieves higher performance while remaining energy efficient (PEs having lighter colors) compared to the E-CGRA.

Energy Efficiency and Performance – Figure 13 plots the energy efficiency and performance of four different

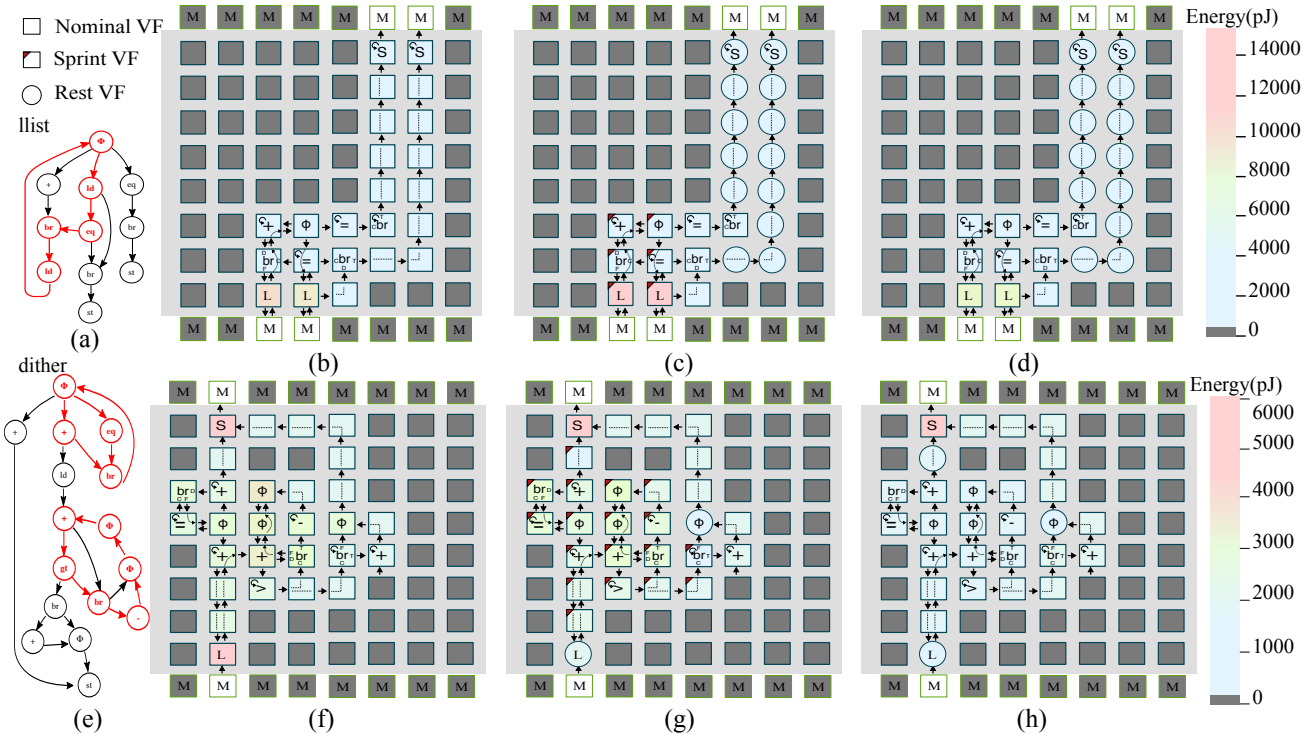


Figure 14. PE Energy Contours for *llist* and *dither* – (a,e) kernel DFGs; (b,f) E-CGRA energy contours; (c,g) UE-CGRA POpt energy contours; (d,h) UE-CGRA EOpt energy contours. Different PE shapes indicate different DVFS states; rest: 0.61 V, 0.33× freq.; nominal: 0.90 V, 1.00× freq.; sprint: 1.23 V, 1.50× freq.. A self-cycle inside a PE indicates one input is constant. Dashed lines indicate bypass paths in a PE. C: condition input of br. D: data input of br. T: output of br when C is true (optional). F: output of br when C is false (optional).

E-CGRAs (all PEs at lowest rest mode, all PEs at highest sprint mode, and two in-between modes) as well as the two performance/energy-optimized UE-CGRA mappings. Results show that statically resting and sprinting the entire E-CGRA can only achieve either high energy efficiency *or* high performance. On the other hand, the UE-CGRA can achieve *both* with carefully chosen power mappings. For example, *llist* (linked-list search) in Figure 14(c) achieves 1.49× performance over the E-CGRA by accelerating critical PEs with the same energy cost. Even in situations where most PEs must sprint for performance (see Figure 14(g)), *dither*'s performance-optimized mapping demonstrates similar energy (1.00×) by resting non-critical PEs.

D. First-Order System-Level Results

This work focuses on detailed evaluation within the CGRA subsystem, but we also take the opportunity to evaluate the CGRA in a small system with a 750 MHz inorder RV32IM core (matching our CGRAs, see Section VI-D) with first-order estimation of reconfiguration and data-loading overheads based on kernel properties (e.g., dataset size). Table III projects the performance and energy efficiency of the processor-CGRA system (with approximate overheads listed) relative to a single core. For example, the linked list kernel (i.e., *llist*) on the UE-CGRA requires 65 cycles for reconfiguration (60 configuration, 3 voltage-scaling, 2 clock-scaling) and 500 cycles to load data before executing for 1000 iterations (and throughput is then measured in iterations/s,

TABLE III. PERFORMANCE AND ENERGY EFFICIENCY FOR INTEGRATED SYSTEM RELATIVE TO PROCESSOR

Mapping	Recurrence		Cost		E-CGRA		EOpt		POpt	
	Ideal	Real	Cfg	Data	Perf	E.Eff	Perf	E.Eff	Perf	E.Eff
llist	5	8	58/65	500	0.94	0.72	0.94	1.02	1.35	0.75
dither	5	8	58/65	500	1.30	0.80	1.30	1.16	1.80	0.93
susan	5	11	58/65	250	1.42	0.55	1.42	0.91	2.10	0.63
fft	4	12	58/65	250	2.31	0.69	2.31	1.53	3.38	1.34
bf	12	24	58/65	8	1.36	0.64	1.18	0.80	1.90	0.64

Ideal = ideal recurrence length (theoretical lower bound) (cycles). Real = actual recurrence length in E-CGRA simulation (cycles). Cfg = cycles to transfer config bits and configure E-CGRA/UE-CGRA. Data = cycles to transfer data. EOpt/POpt = performance/energy-optimized UE-CGRAs. See Table II and Section VI for methodology. Performance and efficiency relative to single in-order RV32IM core.

see Section VI-C). The iteration count amortizes overheads, and modern CGRA compilers typically identify small 10-instruction regions that can be reused 10K+ times [18, 19]).

Next we note from the recurrence columns that our compiler for the baseline E-CGRA is not perfectly optimized (and is indeed not the focus of this work). For example, the theoretical lower bound for the recurrence in *bf* is 12 cycles, but our mapping completes a cycle in 24 cycles. This means that our CGRA performance is conservative and could be higher.

Next we consider the E-CGRA results (with overheads) relative to the core. The most conspicuous impact of true-dependency bottlenecks is the simple *llist* kernel, which per-

forms no better on the E-CGRA (0.94×) than on the core. The *dither* kernel is similar. Performance on other kernels comes from exploiting ILP (up to 2.31×), and energy is worse than the core because many PEs are left active to route data.

Finally, the UE-CGRA results show that **true-dependency bottlenecks can be overcome**. The *llist* kernel improves 1.35× in performance despite its true dependency, and *dither* improves 1.80×. For all other kernels, there is at least one power-mapping approach (either EOpt or POpt) that has good energy efficiency or performs well (e.g., *fft* exploits both ILP and fine-grain DVFS for 3.38× speedup or 1.53× energy).

VIII. Discussion

In this section we discuss various topics about the UE-CGRA design space and irregular loop specialization.

A. Q&A: Scalability of UE-CGRAs?

As CGRAs (both inelastic and elastic) scale in PE count, their area and therefore clock-distribution overhead increases, and we expect VLSI architects to eventually consider classic GALS approaches to reduce the overhead by partitioning the design into synchronous islands (e.g., ETH [20], NVIDIA [31]). A ratiochronous design approach is still a synchronous one, and therefore a UE-CGRA is constrained to live entirely within these synchronous islands.

From an efficiency perspective, we expect a large UE-CGRA island to show similar overheads compared to a large E-CGRA island, with trends comparable to Table I (despite replicating three global clock networks). This is because hierarchical global clock-network gating is very effective for UE-CGRAs. Each PE only selects one of the three clocks, leaving most of the clock network gateable across the array. The compiler can statically analyze for optimal global clock gating because it is aware of all PE clock selections. In addition, PEs on the same clock are naturally clustered due to the producer-consumer relationship in dataflow (e.g., see Figure 14(c,g)), which increases the opportunity to gate large portions of the clock network.

B. Q&A: Performance compared to an out-of-order core?

Out-of-order cores and CGRAs (including UE-CGRAs) are each designed to extract ILP for performance. The mechanisms in the out-of-order core do not typically help to also accelerate irregular loops with true dependencies. In particular, speculation mechanisms target control flow but do not speculate data. Although value prediction has been explored [36], large general-purpose cores tend to prioritize resources for dynamically extracting ILP, resulting in an overall inefficient architecture. Note that an out-of-order core can also be sprinted (monolithically) but it would transition more slowly (e.g., 100s of cycles, [33]) while greatly exacerbating the inefficiencies. If such a core were added to Table III, the results might match UE-CGRA POpt performance but only run at 0.05× efficiency.

C. Q&A: Area efficiency and unrolling for more parallelism?

Utilization is a challenge for CGRAs in general. In practice, the CGRA fabric size is typically tuned such that the largest target workload has a high utilization, but this means that smaller kernels will underutilize CGRA resources (e.g., see Figure 14, our average is 65% utilized). While not implemented in this work, the utilization challenge can be mitigated by unrolling the outer loop and then instantiating multiple instances of the kernel onto different parts of the fabric. Another approach could also launch multiple instances from different kernels onto the fabric in parallel. From the perspective of this work, it is worth noting that the benefits of a UE-CGRA are *complementary* by applying DVFS intra-kernel, so each DFG improves as shown in Figure 13.

IX. Related Work

Reconfigurable spatial architectures including CGRAs are an active area of research [1, 12, 13, 15, 18, 19, 39, 40, 44–47, 53, 59, 61]. Well-known CGRAs include ADRES [40], RaPiD [12], and MorphoSys [53]. More recently, less traditional CGRAs have been explored including the DySER architecture [18, 19], the triggered instructions paradigm [45]. and CGRAs targeting vision, object detection, and learning [1, 13, 59, 61], including in industry [61].

Fine-grain DVFS has been studied with multi-rail voltage supplies [11, 42, 57] and with fully integrated voltage regulators [14, 16, 17, 25, 33, 35, 54, 56]). Most recent work has explored fine-grain DVFS at the core granularity (as compared to simple PEs in this work), where per-domain regulators and PLLs might still be practical. Kim et al. conducted a system-level analysis of per-core DVFS primarily to save energy [33], while Godycki et al. and Torng et al. tried to also improve multicore performance [17, 56]. Truong et al. built a 167-processor array with per-core DVFS and provided a detailed account of multi-rail supplies with power switches [57]. Industry interest in per-core DVFS has also risen [28, 29, 38].

Relatively fewer works explore fine-grain DVFS at the PE level. Semeraro et al. [51] explored fine-grain DVFS across subunits of an out-of-order processor. They assumed asynchronous FIFOs and per-domain PLLs, both of which are likely to be intractable for CGRAs with very small domains. Muramatsu et al. [43] explored fine-grain DVFS for a RISC-V core partitioned into a 6×7 grid. Some isolated work explored inelastic CGRAs [23, 24] but relied on brute-force synchronizers for clock-domain crossings. The high latency is inappropriate for high-performance dataflow (Section IV-B), and synchronizers prevent the use of commercial STA.

There is a long history of research on GALS systems [8, 20, 22, 55]. The most well-known GALS implementations were based on a pausible-clocking scheme with per-domain ring oscillators. The chips built at ETH Zurich were particularly well-known for this approach [20]. Intel has experimented with a mesochronous GALS chip with 80 PEs running at the same frequency but with unknown phase [58]. More recently, NVIDIA's research team designed a spatial accelerator with GALS domains enabled by pausible bisynchronous fifos [31].

The ratiochronous family was initially proposed by Sarmenta et al. [50]. Chabloz et al. continued exploring rational clocking as an alternative to fully asynchronous GALS [3–7]. These works focused on the interface and communication mechanisms rather than on architectural implications. Yadav et al. [62] proposed phase-aligned ratiochronous blocks with fine-grain DVFS. However, they focused on individual circuits in isolation and assumed asynchronous FIFOs for correct interfacing. The RIRI scheme generates rational clocks across phase-aligned boundaries [60], but they do not handle communication microprotocols.

The UE-CGRA compiler power-mapping problem resembles traditional DVFS mode assignment for task graphs mapped onto multiprocessor SoCs [9, 52], but node communication is typically assumed to be free (not true for CGRAs).

X. Conclusion

CGRAs are well-known to be efficient for regular kernels, but once integrated into an SoC they remain idle for irregular workloads. This work explored the potential of UE-CGRAs for enabling the CGRA to also efficiently execute irregular loops. We demonstrated that new techniques co-designed across the compiler, architecture, and VLSI can enable sprinting through true-dependency bottlenecks while also resting to save energy. We also showed that careful VLSI design with ratiochronous clocking can enable robust clock-domain crossings that are fully verifiable with commercial STA tools. Overall, UE-CGRAs show promise in mitigating all three challenges for irregular loop specialization, making CGRAs stronger candidates for specialized compute.

XI. Acknowledgments

This work was supported in part by DARPA SDH Award #FA8650-18-2-7863, DARPA POSH Award #FA8650-18-2-7852, NSF CRI Award #1512937, a research gift from Xilinx, Inc., and the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA, as well as equipment, tool, and/or physical IP donations from Intel, Synopsys, Cadence, and ARM. The authors acknowledge and thank Shunning Jiang for his help in developing the PyMTL3 CGRA model. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL and DARPA or the U.S. Government.

References

[1] I. Bae, B. Harris, H. Min, and B. Egger. Auto-Tuning CNNs for Coarse-Grained Reconfigurable Array-Based Accelerators. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Jul 2018.

[2] R. Bahr et al. Creating an Agile Hardware Design Flow. *Design Automation Conf. (DAC)*, Jul 2020.

[3] J.-M. Chabloz. *Globally-Ratiochronous, Locally-Synchronous Systems*. Ph.D. Thesis, Electronic and Computer Systems Department, KTH, 2012.

[4] J.-M. Chabloz and A. Hemani. A flexible communication scheme for rationally-related clock frequencies. *Int'l Conf. on Computer Design (ICCD)*, Oct 2009.

[5] J.-M. Chabloz and A. Hemani. Distributed DVFS Using Rationally-Related Frequencies and Discrete Voltage Levels. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Aug 2010.

[6] J.-M. Chabloz and A. Hemani. Lowering the latency of interfaces for rationally-related frequencies. *Int'l Conf. on Computer Design (ICCD)*, Oct 2010.

[7] J.-M. Chabloz and A. Hemani. Low-Latency Maximal-Throughput Communication Interfaces for Rationally Related Clock Domains. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, Apr 2013.

[8] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. Ph.D. Thesis, Computer Science Department, Stanford, 1984.

[9] G. Chen, K. Huang, and A. Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *IEEE Trans. on Embedded Computing Systems (TECS)*, Mar 2014.

[10] C. E. Cummings. Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog. *Synopsys Users Group (SNUG)*, 2008.

[11] R. G. Dreslinski. *Near-Threshold Computing: From Single-Core to Many-Core Energy-Efficient Architectures*. Ph.D. Thesis, EECS Department, University of Michigan, 2011.

[12] C. Ebeling, D. C. Cronquist, and P. Franklin. RaPiD: Reconfigurable Pipelined Datapath. *Int'l Conf. on Field Programmable Logic (FPL)*, Sep 1996.

[13] X. Fan, D. Wu, W. Cao, W. Luk, and L. Wang. Stream Processing Dual-Track CGRA for Object Inference. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, Feb 2018.

[14] E. Fluhr et al. The 12-Core POWER8 Processor With 7.6 Tb/s IO Bandwidth, Integrated Voltage Regulation, and Resonant Clocking. *IEEE Journal of Solid-State Circuits (JSSC)*, 50(1):10–23, Jan 2015.

[15] M. Gao and C. Kozyrakis. HRL - Efficient and flexible reconfigurable logic for near-data processing. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Apr 2016.

[16] H. Ghasemi, A. Sinkar, M. Schulte, and N. S. Kim. Cost-Effective Power Delivery to Support Per-Core Voltage Domains for Power-Constrained Processors. *Design Automation Conf. (DAC)*, Jun 2012.

[17] W. Godycki, C. Torng, I. Bukreyev, A. Apsel, and C. Batten. Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2014.

[18] V. Govindaraju et al. DySER: Unifying Functionality and Parallelism Specialization for Energy Efficient Computing. *IEEE Micro*, 33(5), Sep/Oct 2012.

[19] V. Govindaraju, C.-H. Ho, and K. Sankaralingam. Dynamically Specialized Datapaths for Energy-Efficient Computing. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2011.

[20] F. K. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner. GALS at ETH Zurich: success or failure? *Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, Mar 2006.

[21] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu. Elastic CGRAs. *Int'l Symp. on Field Programmable Gate Arrays (FPGA)*, Feb 2013.

[22] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. *Int'l Symp. on Computer Architecture (ISCA)*, May 2002.

[23] S. M. A. H. Jafri et al. Energy-aware coarse-grained reconfigurable architectures using dynamically reconfigurable isolation cells. *Int'l Symp. on Quality Electronic Design (ISQED)*, Mar 2013.

[24] S. M. A. H. Jafri et al. Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in CGRAs. *Int'l Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, Jul 2013.

[25] R. Jevtic et al. Per-Core DVFS with Switched-Capacitor Converters for Energy Efficiency in Manycore Processors. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, 23(4):723–730, Apr 2015.

[26] S. Jiang, B. Ilbeyi, and C. Batten. Mamba: closing the performance gap in productive hardware development frameworks. *Design Automation Conf. (DAC)*, Jun 2018.

[27] S. Jiang, P. Pan, Y. Ou, and C. Batten. PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification. *IEEE Micro*, Jul 2020.

[28] D. Kanter. Haswell's FIVR Extends Battery Life. *Microprocessor Report, The Linley Group*, Jun 2013.

[29] D. Kanter. Ryzen Returns AMD to the Desktop. Microprocessor Report, The Linley Group, Mar 2017. `http://www.linleygroup.com/mpr/article.php?id=11775`.

[30] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh. HyCUBE: A CGRA with Reconfigurable Single-cycle Multi-hop Interconnect. *Design Automation Conf. (DAC)*, Jun 2017.

[31] B. Khailany et al. A Modular Digital VLSI Flow for High-Productivity SoC Design. *Design Automation Conf. (DAC)*, Jun 2018.

[32] M. Khazraee, L. Zhang, L. Vega, and M. B. Taylor. Moonwalk: NRE Optimization in ASIC Clouds. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2017.

[33] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System-Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2008.

[34] M. Lanuzza, P. Corsonello, and S. Perri. Fast and Wide Range Voltage Conversion in Multisupply Voltage Designs. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, Feb 2015.

[35] Y. Lee et al. POWER7: IBM's Next Generation POWER Microprocessor. *Symp. on High Performance Chips (Hot Chips)*, Aug 2015.

[36] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value Locality and Load Value Prediction. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Sep 1996.

[37] D. Lockhart, G. Zibrat, and C. Batten. PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2014.

[38] A. J. Martinez et al. Haswell: The Fourth-Generation Intel Core Processor. *IEEE Micro*, 34(2):6–20, Mar/Apr 2014.

[39] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. *IEEE Proceedings Computers and Digital Techniques*, Nov 2003.

[40] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix. *Int'l Conf. on Field Programmable Logic (FPL)*, Sep 2003.

[41] G. Michelogiannakis, J. Balfour, and W. J. Dally. Elastic-buffer flow control for on-chip networks. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2009.

[42] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu. Booster: Reactive Core Acceleration For Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2012.

[43] A. Muramatsu et al. 12% Power reduction by within-functional-block fine-grained adaptive dual supply voltage control in logic circuits with 42 voltage domains. *European Solid-State Circuits Conf. (ESSCIRC)*, Sep 2011.

[44] T. Oh, B. Egger, H. Park, and S. Mahlke. Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. *International Conference on Languages, Compilers, Tools, and Theory for Embedded Systems (LCTES)*, Jun 2009.

[45] A. Parashar et al. Triggered Instructions: A Control Paradigm for Spatially-programmed Architectures. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2013.

[46] Y. Park, H. Park, and S. Mahlke. CGRA express: accelerating execution using dynamic operation fusion. *Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Oct 2009.

[47] M. Pellauer et al. Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2019.

[48] C. Petty and P. Shockman. Odd Number Divide By Counters With 50% Outputs and Synchronous Clocks. Application Note, 1999.

[49] G. S. Ravi and M. H. Lipasti. CHARSTAR: Clock Hierarchy Aware Resource Scaling in Tiled Architectures. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2017.

[50] L. F. G. Sarmenta, G. A. Pratt, and S. A. Ward. Rational clocking. *Int'l Conf. on Computer Design (ICCD)*, Oct 1995.

[51] G. Semeraro et al. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. *Int'l Symp. on Microarchitecture (MICRO)*, Nov 2002.

[52] A. K. Singh, A. Das, and A. Kumar. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. *Design Automation Conf. (DAC)*, Jun 2013.

[53] H. Singh et al. MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. *IEEE Transactions on Computers*, May 2000.

[54] A. A. Sinkar, H. R. Ghasemi, M. J. Schulte, U. R. Karpuzcu, and N. S. Kim. Low-Cost Per-Core Voltage Domain Support for Power-Constrained High-Performance Processors. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, 22(4):747–758, Apr 2014.

[55] P. Teehan, M. Greenstreet, and G. Lemieux. A Survey and Taxonomy of GALS Design Styles. *IEEE Design and Test of Computers*, Oct 2007.

[56] C. Torng, M. Wang, and C. Batten. Asymmetry-Aware Work-Stealing Runtimes. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2016.

[57] D. Truong et al. A 167-Processor Computational Platform in 65-nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4):1130–1144, Apr 2009.

[58] S. Vangali et al. 80-Tile 1.28 TFlops Network-on-Chip in 65 nm CMOS. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2007.

[59] A. Vasilyev et al. Evaluating programmable architectures for imaging and vision applications. *Int'l Symp. on Microarchitecture (MICRO)*, Oct 2016.

[60] R. Wang, H. Wang, B. Fan, and L. Yang. RIRI scheme: A robust instant-responding ratiochronous interface with zero-latency penalty. *Int'l Conf. on Circuits and Systems (ISCAS)*, May 2011.

[61] A Coarse Grain Reconfigurable Array (CGRA) for Statically Scheduled Data Flow Computing. WAVE Computing White Paper, 2018. `https://wavecomp.ai/wp-content/uploads/2018/12/WP_CGRA.pdf`.

[62] M. K. Yadav, M. R. Casu, and M. Zamboni. DVFS Based on Voltage Dithering and Clock Scheduling for GALS Systems. *Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2012.