SCALING A RECONFIGURABLE DATAFLOW ACCELERATOR

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Yaqi Zhang
April 2020

Abstract

This thesis tells you all you need to know about life, the universe, and everything.

Acknowledgements

I would like to thank my mother and the little green men from Mars.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Chapter 2

Background

2.1 Plasticine

2.2 Spatial

We use Spatial, an open source domain specific language for reconfigurable accelerators, to target spatial architectures [**?**]. Spatial describes applications with nested loops and an explicit memory hierarchy that captures data movement on-chip and off-chip. This exposes design parameters that are essential for achieving high performance on spatial architectures, including blocking size, loop unrolling factors, inner-loop pipelining, and coarse-grained pipelining of arbitrarily nested loops. To enable loop-level parallelization and pipelining, Spatial automatically banks and buffers intermediate memories between loops. An example of outer product—element-wise multiplication of two vectors resulting in a matrix—in Spatial is shown in Figure 2.1. For spatial architectures, Design Space Exploration (DSE) of parameters (e.g., *op1*, *op2*, *ip*, *tsA*, *tsB*) is critical to achieve good resource utilization and performance [**?**].

```
// Host to accelerator register for scalar input with
// user annotated value
val N = ArgIn[Int]; bound(N) = 1024
// 1-D DRAM size in N
val vecA, vecB = DRAM[T](N)
// 2-D DRAM size in NxN
val matC = DRAM[T](N, N)
// Loop unrolling factors
val op1, op2, ip:Int = ...
// Blocking sizes of vecA and vecB
val tsA, tsB:Int = ...

// Accelerator kernel
C0: Accel {
  // C1 is parallelized by op1
  C1: Foreach(min=0, step=tsA, max=N, par=op1){ i =>
    // Allocate 1-D scratchpad size in tsA
    val tileA = SRAM[T](tsA)
    // Load range i to i+tsA of vectorA from off- to
    // on-chip parallelized by ip
    C2: tileA load vecA(i::i+tsA par ip)
    C3: Foreach(min=0, step=tsB, max=N, par=op2) { j =>
      val tileB = SRAM[T](tsB)
      C4: tileB load vecB(j::j+tsB par ip)
      // 2-D scratchpad
      val tileC = SRAM[T](tsA, tsB)
      C5: Foreach(min=0, step=1, max=tsA){ ii =>
        Foreach(min=0, step=1, max=tsB, par=ip) { jj =>
          tileC(ii, jj) = tileA(ii) * tileB(jj)
        }
      }
      // Store partial results to DRAM
      C6: matC(i::i+tsA, j::j+tsB par ip) store tileC
    }
  }
}
```

Figure 2.1: Example of outer product in Spatial pseudocode.

Chapter 3

Architecture

3.1 Plasticine Specialization for RNN Serving

To show efficient execution of the loop and parallel pattern constructs, we map our implementation onto a spatial architecture, Plasticine. **Foreach** at Line 17, 19 and **Reduce** at Line 22, 23 are mapped to PCUs on Plasticine. When the application size is small, these constructs are executed using pipelined SIMD lanes within a single PCU. When the application size is large, multiple PCUs can be used to parallelize and pipeline the dot product across PCUs. Element-wise operations can be executed in a deep pipeline formed by chaining multiple PCUs.

To fit an RNN's weights on-chip, we execute our application with low-precision arithmetics. In this section, we propose the necessary micro-architectural changes to support low-precision arithmetics on Plasticine. We also discuss architectural parameter selection for Plasticine to serve RNN applications efficiently.

3.1.1 Mixed-Precision Support

Previous works [**?**, **?**] have shown that low-precision inference can deliver promising performance improvements without sacrificing accuracy. In the context of reconfigurable architectures such as FPGAs, low-precision inference not only increases compute density, but also reduces required on-chip capacity for storing weights and intermediate data.

To support low-precision arithmetics without sacrificing coarse-grained reconfigurability, we introduce two low-precision struct types in Spatial: a tuple of 4 8-bit and 2 16-bit floating-point numbers, `4-float8` and `2-float16` respectively. Both types packs multiple low-precision values into a single precision storage. We support only 8 and 16-bit precisions, which are commonly seen in deep learning inference hardwares. Users can only access values that are 32-bit aligned. This constraint guarantees that the microarchitectual change is only local to the PCU. Banking and DRAM access granularity remains intact from the original design.

Figure 3.1 (a) shows the original SIMD pipeline in a Plasticine PCU. Each FU supports both floating-point and fix-point operations. When mapping applications on Plasticine, the inner most loop body is vectorized across the lanes of the SIMD

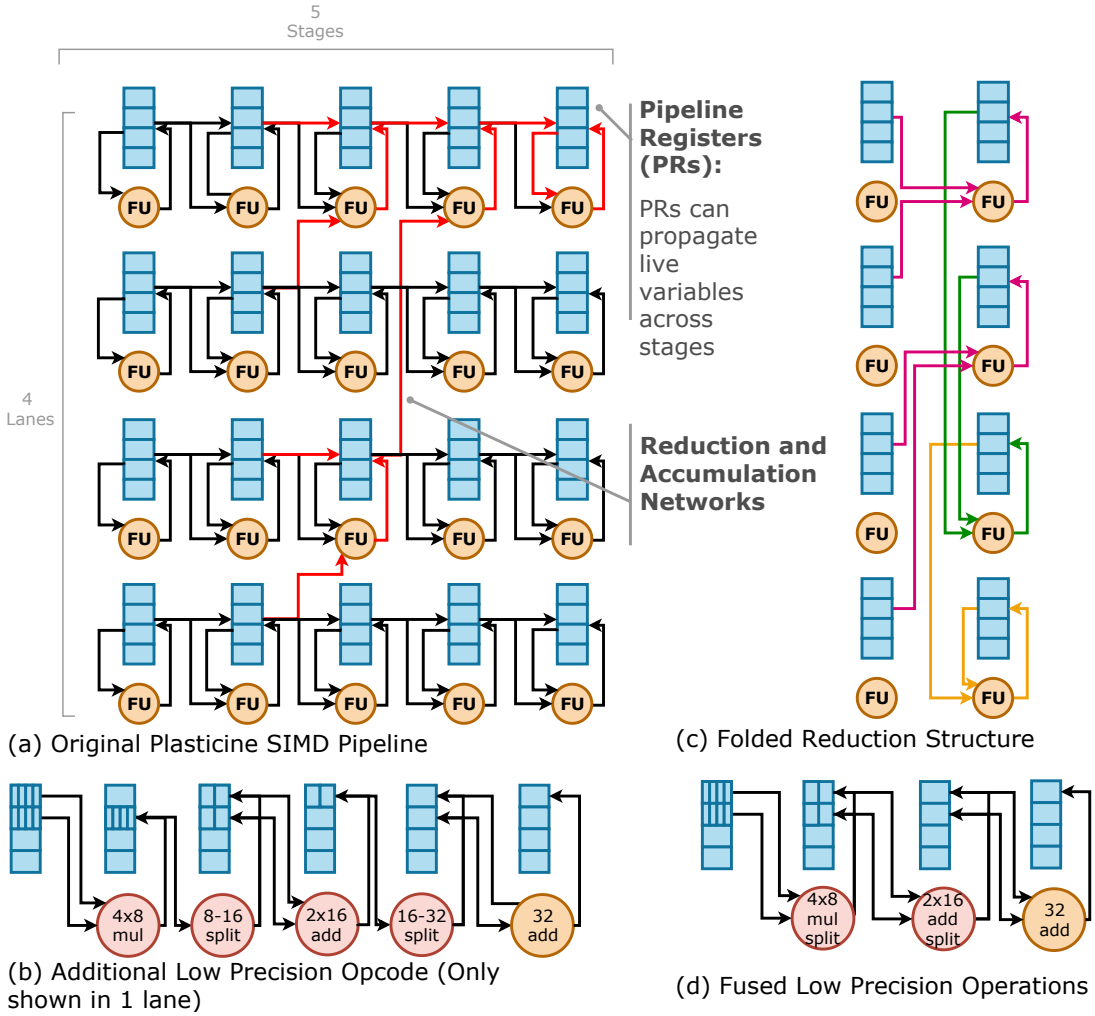


Figure 3.1: Plasticine PCU SIMD pipeline and low-precision support. Red circles are the new operations. Yellow circles are the original opertaions in Plasticine. In (d) the first stage is fused $1^{st}, 2^{nd}$ stages, and the second stage is fused 3^{nd}, 4^{th} stages of (b).

pipeline, and different operations of the loop body are mapped to different stages. Each pipeline stage contains a few pipeline registers (PRs) that allow propagation of live variables across stages. Special cross-lane connections as shown in red in Figure 3.1 enable reduction operations. To support 8-bit element-wise multiplication and 16-bit reduction, we add 4 opcodes to the FU, shown in Figure 3.1 (b). The 1^{st} and 3^{rd} stages are element-wise, low-precision operations that multiply and add 4 8-bit and 2 16-bit values, respectively. The 2^{nd} and 4^{th} stages rearrange low-precision values into two registers, and then pad them to higher precisions. The 5^{th} stage reduces the two 32-bit value to a single 32-bit value using the existing add operation. From here, we can use the original reduction network shown in Figure 3.1 (a) to complete the remaining reduction and accumulates in 32-bit connection.

With 4 lanes and 5 stages, a PCU first reads 16 8-bit values, performs 8-bit multiplication followed by rearrangement and padding, and then produce 16 16-bit values after the second stage. The intermediate values are stored in 2 PRs per lane. Next, 16 16-bit values are reduced to 8 16-bit values and then rearranged to 8 32-bit value in 2 PRs per lane. Then, the element-wise addition in 32-bit value reduces the two registers in each line into 4 32-bit values. These values are fed through the reduction network that completes the remaining reduction and accumulation in two plus one stages.

In a more aggressive specialization, we can fuse the multiply and rearange into the same stage. We also fuse the first low-precision reduction with the next rearange as shown in Figure 3.1 (d). In this way, we can perform the entire low-precision map-reduce in 2 stages in addition to the original full precision reduction. In order to maximize hardware reuse, we assume that it is possible to construct a full precision FU using low-precision FUs. In addition, we observe that the original reduction network in the SIMD lanes could lead to low FU utilization. To improve FU utilization, we fold the entire tree structure in a single stage. Figure 3.1 (c) shows the folded reduction accumulation structure. Specifically, latter reductions in the tree are mapped to earlier stages in the pipeline. In this setup, the entire reduction plus accumulation is still fully pipelined in $\log_2(\#_{LANE}) + 1$ cycles with no structural hazard. With fused reduced-precision multiplication and reduction, and folded reduction tree, a PCU is

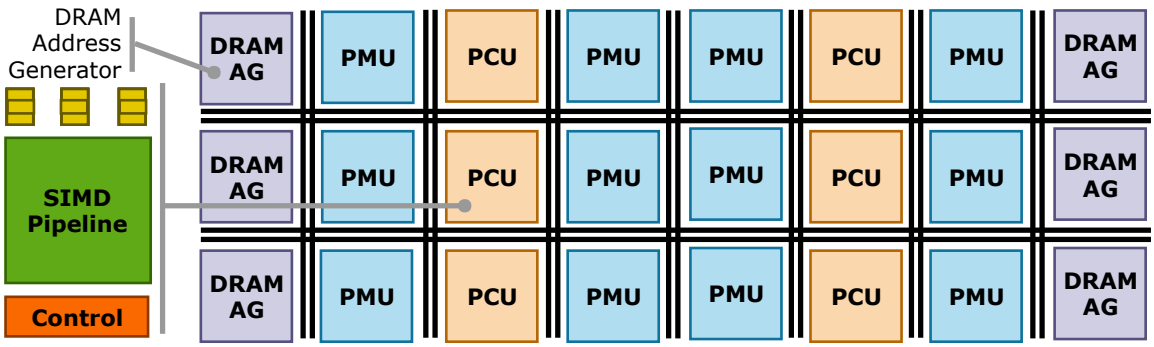


Figure 3.2: Variant configuration of Plasticine for serving RNN.

able to perform all map-reduce that accumulates $4\#_{LANE}$ 8-bit values using 4 stages. All the operations are completed in $2 + \log_2(\#_{LANE}) + 1$ cycles.

3.1.2 Sizing Plasticine for RNN Serving

Evaluating an RNN cell containing N hidden units and N input features requires $2N^2$ computations and $N^2 + N$ memory reads. With large N, the compute to memory ratio is 2:1. The original Plasticine architecture uses a checkerboard layout with 1 to 1 ratio between PCU and PMU. A PCU has 6 stages and 16 lanes, and a PMU has 16 banks. This provides a 6:1 ratio between compute resource and on-chip memory read bandwidth. As a result of this layout, on-chip memory read bandwidth becomes the bottleneck for accelerating RNN serving applications. Given that RNNs cover a wide range of important applications, we select a Plasticine configuration tailored for RNN serving. Specifically, we choose a 2 to 1 PMU-PCU ratio with 4 stages in each PCU. Figure 3.2 shows the layout of this Plasticine variant.

3.2 On-chip Network

This section discusses communication characteristics common in applications that have been spatially mapped to CGRAs. Because CGRAs encompass a broad range of architectures, we first describe the abstract machine model of our target CGRA for this study, shown in Figure 3.2. The CGRA contains Physical Blocks (PBs) corresponding to distributed hardware resources, including compute units, scratchpads, and DRAM controllers. The communication between PBs, sent over a reconfigurable network, is purely streaming. Compute PBs have a simple control mechanism: they wait on input data dependencies and stall for backpressure from the network. The network guarantees exactly-once, in-order delivery with variable latency, and communication between PBs can have varying granularities (e.g., 512-bit vector or 32-bit scalar).

In this study, we focus on two categories of CGRA architectures. The first architecture uses pipelining in compute PBs, as shown in Figure 3.2. To provide high throughput, each stage of the pipeline exploits SIMD parallelism, and multiple SIMD operations are pipelined within a PB. Plasticine, a recently proposed CGRA, is an example of a pipelined architecture [**?**].

The second architecture uses time-scheduled execution, where each PB executes a small loop of instructions (e.g., 6) repeatedly. The scheduling window is small enough that instructions are stored as part of the configuration fabric, without dynamic instruction fetch overhead. This execution model creates more interleaved pipelining across PBs with communication that is tolerant of lower network throughput, which provides an opportunity to share links. Many proposed CGRAs and domain-specific architectures use this *time-scheduled* form of computation, including Brainwave [**?**] and DaDianNao [**?**].

3.2.1 Application Characteristics

The requirements of an interconnection network are a function of the communication pattern of the application, underlying CGRA architecture, and compilation process. We identify the following key characteristics of spatially mapped applications:

Vectorized communication

Recent hardware accelerators use large-granularity compute tiles (e.g., vectorized compute units and SIMD pipelines) for SIMD parallelism [**?**, **?**], which improves compute density while minimizing control and configuration overhead. Coarser-grained computation typically increases the size of communication, but glue logic, reductions, and loops with carried dependencies (i.e., non-parallelizable loops) contribute to scalar communications. This variation in communication motivates specialization for optimal area- and energy-efficiency: separate networks for different communication granularities.

Broadcast and incast communication

A key optimization for spatial reconfigurable accelerators is the parallelization of execution across PBs. This parallelization involves unrolling outer loop nests in addition to the vectorization of the inner loop. For neural network accelerators, this corresponds to parallelizing one layer across different channels. By default, pipeline parallelism involves one-to-one communication between dependent stages. However, when a consumer stage is parallelized, the producer sends a one-to-many broadcast to all of its consumers. Similarly, when a producer stage is parallelized, all partial results are sent to the consumer, forming a many-to-one incast link. When both the producer and the consumer are parallelized, the worst case is many-to-many communication, because the parallelized producers may dynamically alternate between parallelized receivers.

Compute to memory communication

To encourage better sharing of on-chip memory capacity, many accelerators have shared scratchpads, either distributed throughout the chip or on its periphery [**?**, **?**, **?**]. Because the compute unit has no local memory to buffer temporary results, the results of all computations are sent to memory through the network. This differs from the NoCs used in multi-processors, where each core has a local cache to buffer intermediate results. Studies have shown that for large-scale multi-processor systems,

network latency—not throughput—is the primary performance limiter [**?**]. For spatial accelerators, however, compute performance is limited by network throughput, and latency is comparatively less important.

Communication-aware compilation

Unlike the dynamic communication of multi-processors, communication on spatial architectures is created statically by compiling and mapping the compute graph onto the distributed PB resources. As the compiler performs optimization passes, such as unrolling and banking, it has static knowledge about communication generated by these transformations. This knowledge allows the compiler to accurately determine which network flows in the transformed design correspond to throughput-critical inner-loop traffic and which correspond to low-bandwidth outer-loop traffic.

We select a mix of applications from domains where hardware accelerators have shown promising performance and energy-efficiency benefits, such as linear algebra, databases, and machine learning. Table 3.1 lists the applications and their data size. Figure 3.3 shows, for each design, which resource limits performance: compute, on-chip memory, or DRAM bandwidth. DotProduct, TPCHQ6, OuterProduct, and BlackScholes are DRAM bandwidth-bound applications. These applications use few on-chip resources to achieve maximum performance, resulting in minimal communication. Lattice (a fast inference model for low-dimensional regression [**?**]), GDA, Kmeans, SGD, and LogReg are compute-intensive applications; for these, maximum performance requires using as much parallelization as possible. Finally, LSTM, GRU, and LeNet are applications that are limited by on-chip memory bandwidth or capacity. For compute- and memory-intensive applications, high utilization translates to a large interconnection network bandwidth requirement to sustain application throughput.

Figure 3.4(a,b) shows the communication pattern of applications characterized on the pipelined CGRA architecture, including the variation in communication granularity. Compute and on-chip memory-bound applications show a significant amount of high-bandwidth communication (links with almost 100% activity). A few of these

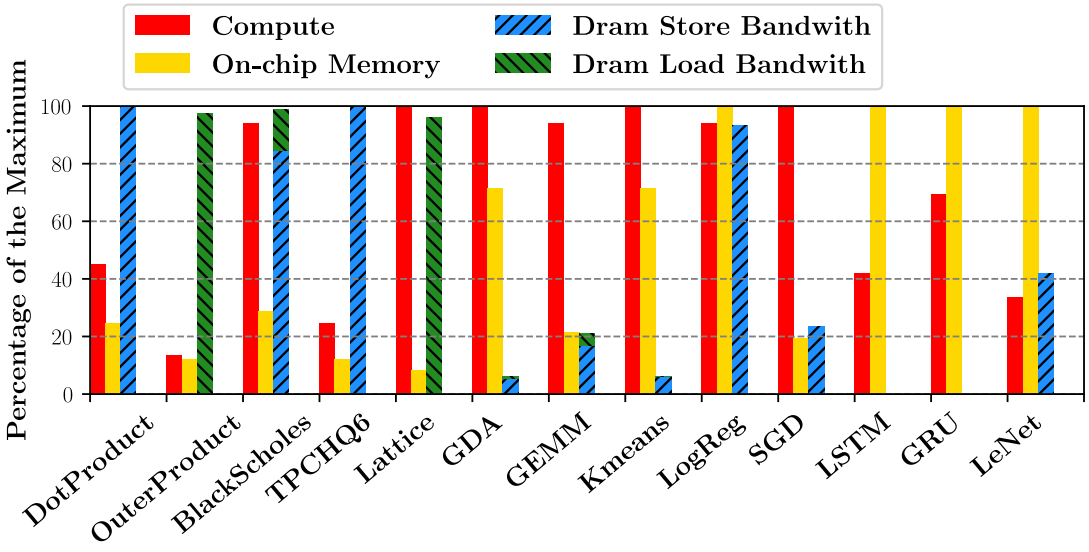
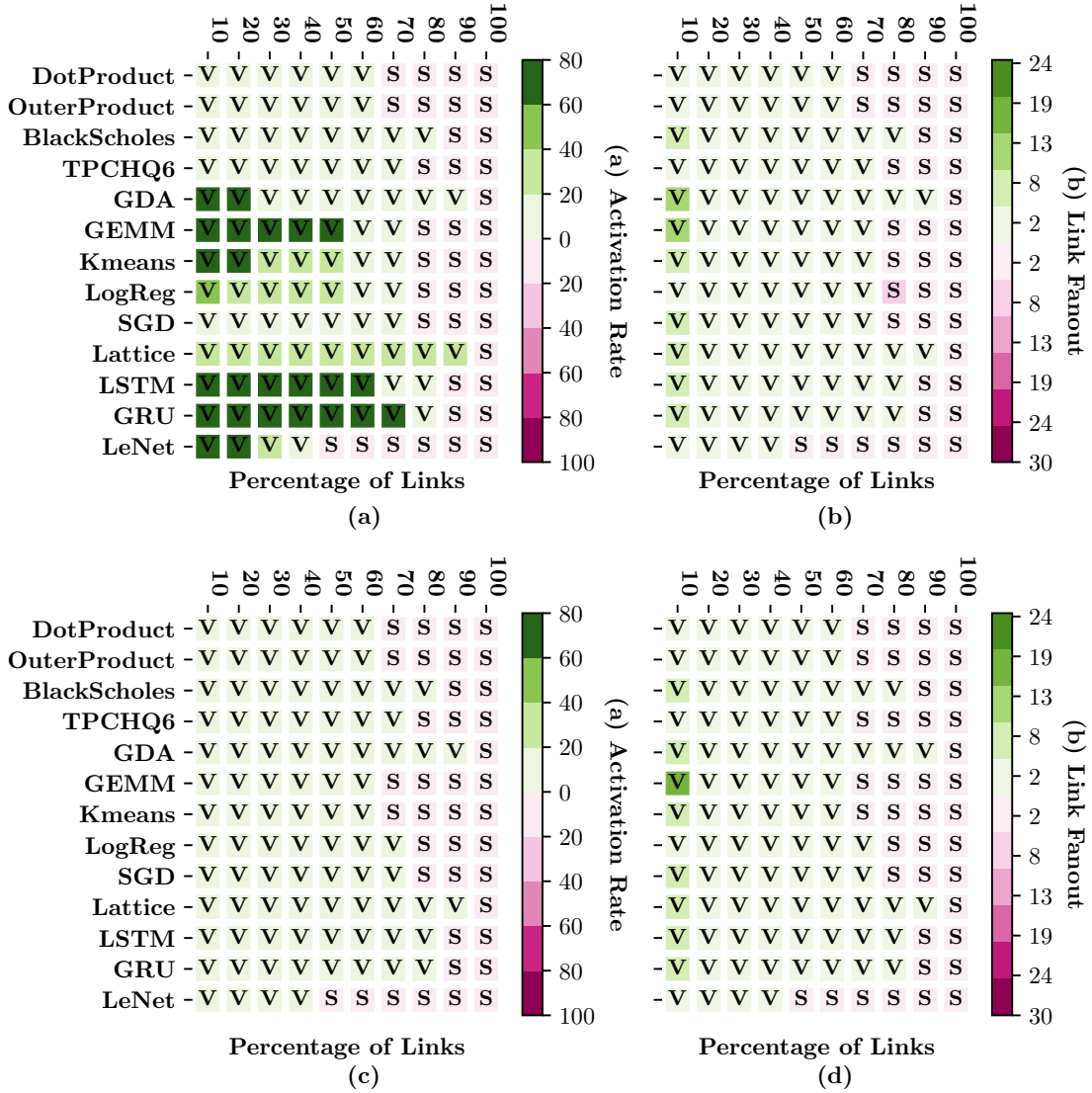


Figure 3.3: Physical resource and bandwidth utilization for various applications.



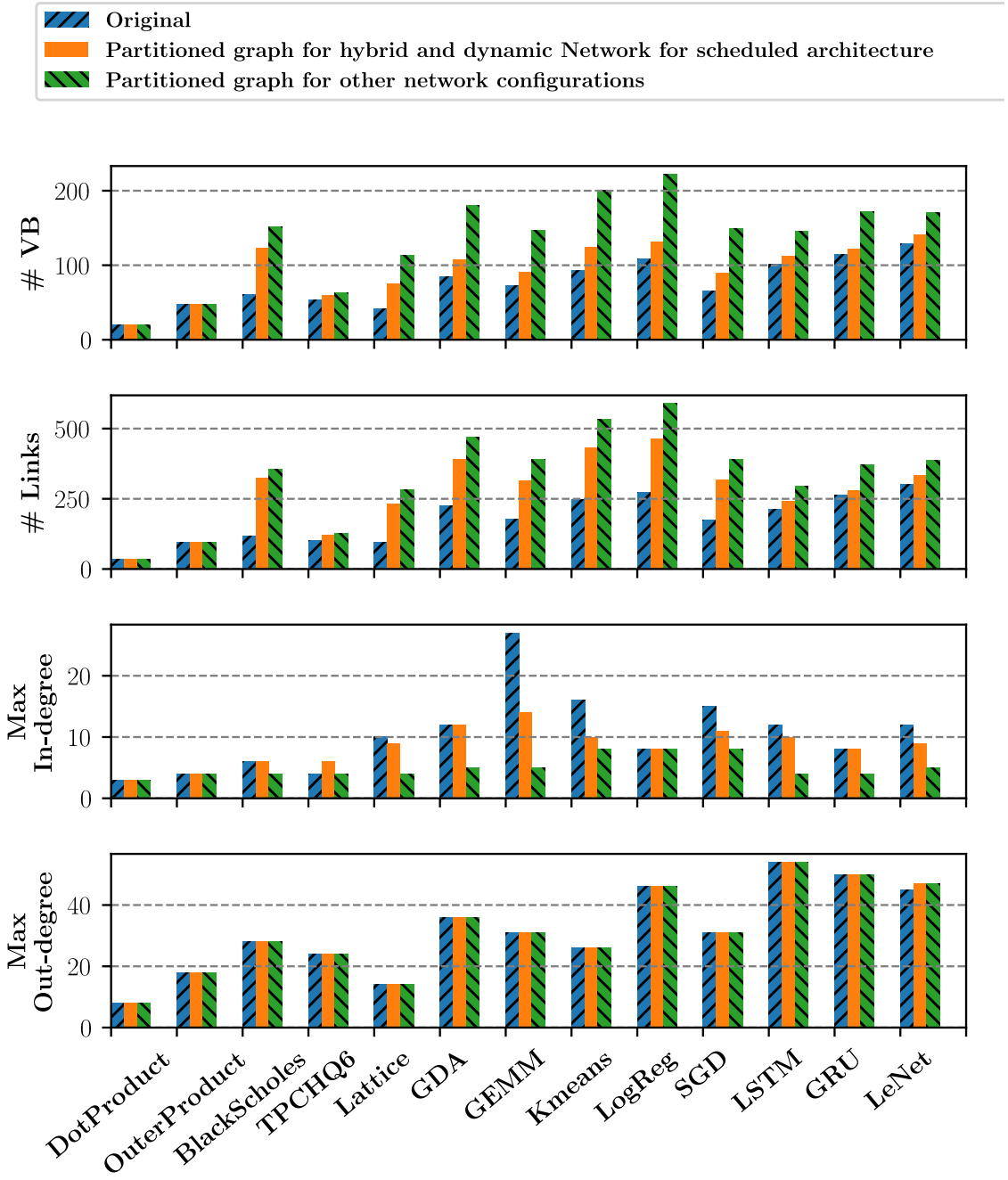


Figure 3.5: Characteristics of program graphs.

high-bandwidth links also exhibit high broadcast fanout. Therefore, a network architecture must provide sufficient bandwidth and efficient broadcasts to sustain program throughput. On the contrary, time-scheduled architectures, shown in Figure 3.4(c,d), exhibit lower bandwidth requirements due to the lower throughput of individual compute PBs. Even applications limited by on-chip resources have less than a 30% firing rate on the busiest logical links; this reveals an opportunity for link sharing without sacrificing performance.

Figure 3.5 shows statistics describing the VB dataflow graph before and after partitioning. The blue bars show the number of VBs, number of logical links, and maximum VB input/output degrees in the original parallelized program; the yellow and green bars show the same statistics after partitioning. Fewer VBs are partitioned for hybrid networks and dynamic networks with the time-scheduled architecture, as explained in Section **??**. The output degree does not change with partitioning because most outputs with a large degree are from broadcast links.

3.2.2 Design Space for Network Architectures

We start with several statically allocated network designs, where each SIMD pipeline connects to several switches, and vary flow control strategies and network bisection bandwidth. In these designs, each switch output connects to exactly one switch input for the duration of the program. We then explore a dynamic network, which sends program data as packets through a NoC. The NoC uses a table-based routing scheme at each router to allow for arbitrary routes and tree-based broadcast routing. Finally, we explore the benefits of specialization by evaluating design points that combine several of these networks to leverage the best features of each.

Static networks

We explore static network design points along three axes. First, we study the impact of flow-control schemes in static switches. In credit-based flow control [**?**], the source and destination PBs coordinate to ensure that the destination buffer does not overflow. For this design point, switches only have a single register at each input,

and there is no backpressure between switches. The alternate design point uses a skid-buffered queue with two entries at each switch; using two entries enables per-hop backpressure and accounts for a one-cycle delay in stalling the upstream switch. At full throughput, the receiver will consume data as it is sent and no queue will ever fill up. The second axis studied is the bandwidth, and therefore routability, of the static network. We vary the number of connections between switches in each direction, which trades off area and energy for bandwidth. Finally, we explore specializing static links: using a separate scalar network to improve routability at a low cost.

Dynamic networks

Our primary alternate design is a dynamic NoC using per-hop virtual channel flow control. Routing and Virtual Channel (VC) assignment are table-based: the compiler performs static routing and VC allocation, and results are loaded as a part of the routers' configurations at runtime. The router has a separable, input-first VC and switch allocator with a single iteration and speculative switch allocation [**?**]. Input buffers are sized just large enough (3 entries) to avoid credit stalls at full throughput. Broadcasts are handled in the network with duplication occurring at the last router possible to minimize energy and congestion. To respect the switch allocator's constraints, each router sends broadcasts to output ports sequentially and in a fixed order. This is because the switch allocator can only grant one output port per input port in every cycle, and the RTL router's allocator does not have sufficient timing slack to add additional functionality. We also explore different flit widths on the dynamic network, with a smaller bus taking multiple cycles to transmit a packet.

Because CGRA networks are streaming—each PB pushes the result to the next PB(s) without explicit request—the network cannot handle routing schemes that may drop packets; otherwise, application data would be lost. Because packet ordering corresponds directly to control flow, it is also imperative that all packets arrive in the order they were sent; this further eliminates adaptive or oblivious routing from consideration. We limit our study of dynamic networks to statically placed and routed source routing due to these architectural constraints. PBs propagate backpressure signals from their outputs to their inputs, so they must be considered as part of

the network graph for deadlock purposes [**?**]. Furthermore, each PB has fixed-size input buffers; these are far too small to perform high-throughput, end-to-end credit-based flow control in the dynamic network for the entire program [**?**]. Practically, this means that no two logical paths may be allowed to conflict at *any* point in the network; to meet this guarantee, VC allocation is performed to ensure that all logical paths traversing the same physical link are placed into separate buffers.

Hybrid networks

Finally, we explore hybrids between static and dynamic networks that run each network in parallel. During static place and route, the highest-bandwidth logical links from the program graph are mapped onto the static network; once the static network is full, further links are mapped to the dynamic network. By using compiler knowledge to identify the relative importance of links—the link fanout and activation factor—hybrid networks can sustain the throughput requirement of most high-activation links while using the dynamic network for low-activation links.

3.2.3 Performance, Area, and Energy Modeling

We use a cycle-accurate simulator to model the pipeline and scheduling delay for the two types of architectures, integrated with DRAMSim [**?**] to model DRAM access latency. For static networks, we model a distance-based delay for both credit-based and per-hop flow control. For dynamic networks, we integrate our simulator with Booksim [**?**], adding support for arbitrary source routing using look-up tables. Finally, to support efficient multi-casting in the dynamic network, we modify Booksim to duplicate broadcast packets at the router where their paths diverge. At the divergence point, the router sends the same flit to multiple output ports over multiple cycles. We assume each packet carries a unique ID that is used to look up the output port and next VC in a statically generated routing table, and that the ID is roughly the same size as an address. When the packet size is greater than the flit size, the transmission of a single packet takes multiple cycles.

Benchmark	**Description**	**Data Size**
DotProduct	Inner product	1048576
OuterProduct	Outer product	1024
BlackScholes	Option pricing	1048576
TPCHQ6	TPC-H query 6	1048576
Lattice	Lattice regression [**?**]	1048576
GDA	Gaussian discriminant analysis	127×1024
GEMM	General matrix multiply	$256 \times 256 \times 256$
Kmeans	K-means clustering	k=64, dim=64, n=8192, iter=2
LogReg	Logistic regression	8192×128, iter=4
SGD	Stochastic gradient descent for a single layer neural network	16384×64, epoch=10
LSTM	Long short term memory recurrent neural network	1 layer, 1024 hidden units, 10 time steps
GRU	Gated recurrent unit recurrent neural network	1 layer, 1024 hidden units, 10 time steps
LeNet	Convolutional neural network for character recognition	1 image

Table 3.1: Benchmark summary

Area and power

To efficiently evaluate large networks, we start by characterizing the area and power consumption of individual routers and switches used in various network configurations. The total area and energy are then aggregated over all switches and routers in a particular network. We use router RTL from the Stanford open source NoC router [**?**] and our own parameterized switch implementation. We synthesize using Synopsys Design Compiler with a 28 nm technology library and clock-gating enabled, meeting timing at a 1 GHz clock frequency. Finally, we use Synopsys PrimeTime to back-annotate RTL signal activity to the post-synthesis switch and router designs to estimate gate-level power.

We found that power consumption can be broken into two types: inactive power consumed when switches and routers are at zero-load (P_{inactive}, which includes both dynamic and static power), and active power. The active power, as shown in Section 3.2.3, is proportional to the amount of data transmitted. Because power scales linearly with the amount of data movement, we model the marginal energy to transmit a single flit of data (flit energy, E_{flit}) by dividing active energy by the number flits transmitted in the testbench:

$$E_{\text{flit}} = \frac{(P - P_{\text{inactive}})\, T_{\text{testbench}}}{\#\text{flit}} \tag{3.1}$$

While simulating an end-to-end application, we track the number of flits transmitted at each switch and router in the network, as well as the number of switches and routers allocated by place and route. We assume unallocated switches and routers are perfectly power-gated, and do not consume energy. The total network energy for an application on a given network (E_{net}) can be computed as:

$$E_{\text{net}} = \sum_{\text{allocated}} P_{\text{inactive}} T_{\text{sim}} + E_{\text{flit}} \#\text{flit}, \tag{3.2}$$

where P_{inactive}, E_{flit}, and #flit are tabulated separately for each network resource.

Figure 3.9 shows that switch and router power scale linearly with the rate of data transmission, but that there is non-zero power at zero-load. For simulation, the duty

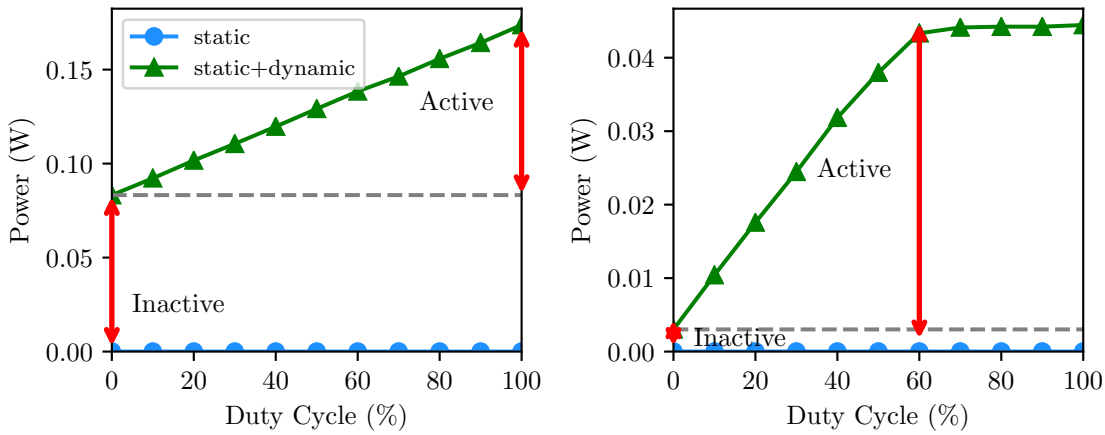


Figure 3.6: Switch and router power with varying duty cycle.

cycle refers to the amount of offered traffic, not accepted traffic. Because our router uses a crossbar without speedup [**?**], the testbench saturates the router at 60% duty cycle when providing uniform random traffic. Nonetheless, router power still scales linearly with accepted traffic.

A sweep of different switch and router parameters is shown in Figure 3.10. Subplots (d,e,f) show the energy necessary to transmit a single bit through a switch or router. Subplot (a) shows the roughly quadratic scaling of switch area with the number of links between adjacent switches. Vector switches scale worse with increasing bandwidth than scalar switches, mostly due to increased crossbar wire load. At the same granularity, a router consumes more energy a switch to transmit a single bit of data, even though the overall router consumes less power (as shown in Figure 3.9); this is because the switch has a higher throughput than the router. The vector router has lower per-bit energy relative to the scalar router because it can amortize the cost of allocation logic, whereas the vector switch has higher per-bit energy relative to the scalar switch due to increased capacitance in the large crossbar. Increasing the number of VCs or buffer depth per VC also significantly increases router area and energy, but reducing the router flit width can significantly reduce router area.

Overall, these results show that scaling static bandwidth is cheaper than scaling

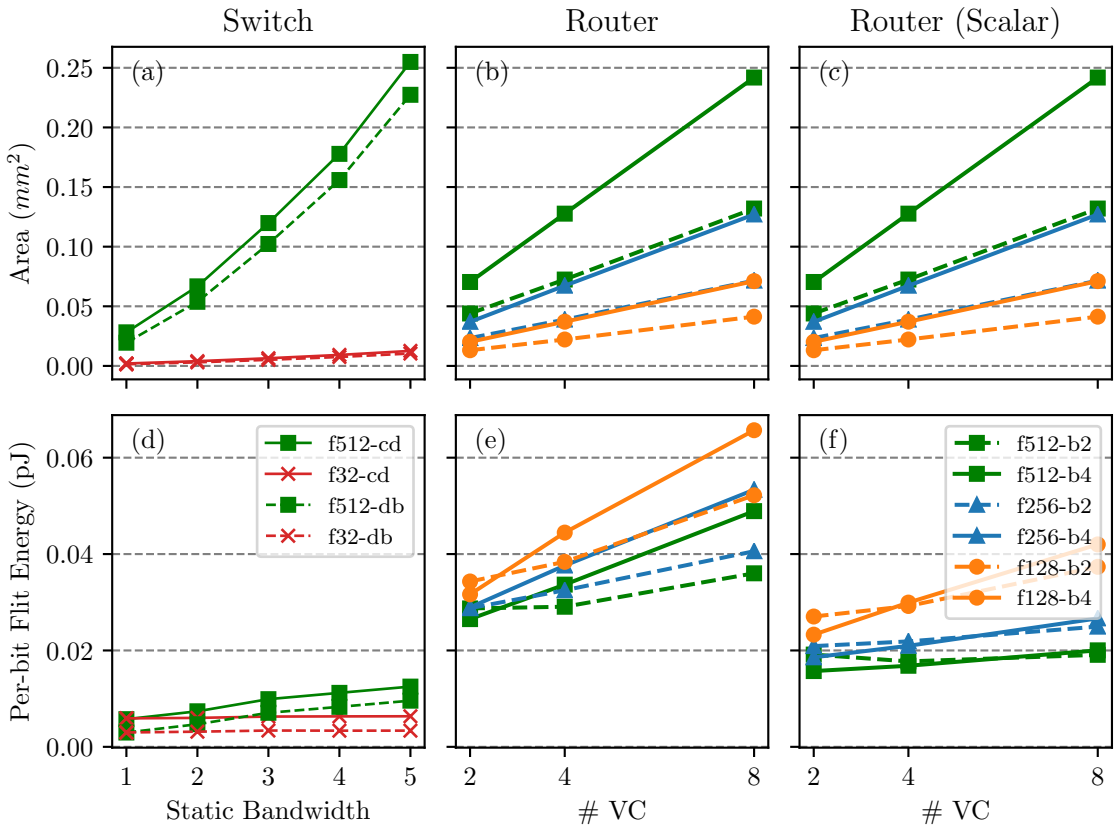


Figure 3.7: Area and per-bit energy for (a,d) switches and (b,c,e,f) routers. (c,f) Subplots (c,f) show area and energy of the vector router when used for scalar values (32-bit).

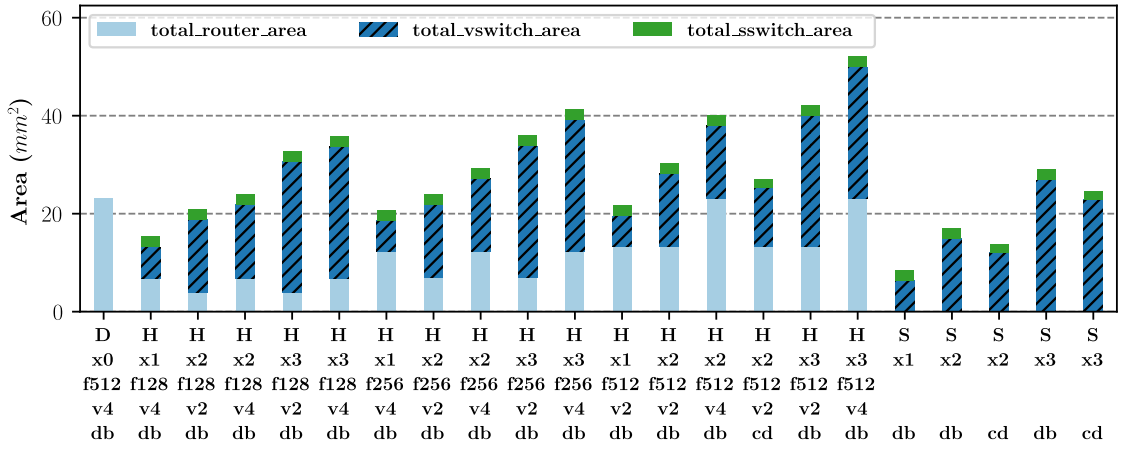


Figure 3.8: Area breakdown for all network configurations.

dynamic bandwidth, and a dynamic network with small routers can be used to improve link sharing for low bandwidth communication. We also see that a specialized scalar network, built with switches, adds negligible area compared to and is more energy efficient than the vector network. Therefore, we use a static scalar network with a bandwidth of 4 for the remainder of our evaluation, except when evaluating the pure dynamic network. The dynamic network is also optimized for the rare instances when the static scalar network is insufficient. When routers transmit scalar data, the high bits of data buffers are clock-gated, reducing energy as shown in (f). Figure 3.11 summarizes the area breakdown of all the network configurations that we evaluate.

3.2.4 Network Architecture Exploration

We evaluate our network configurations in five dimensions: performance (perf), performance per network area (perf/area), performance per network power (perf/watt), network area efficiency (1/area), and network power efficiency (1/power). Among these metrics, performance is the most important: networks only consume a small fraction of the overall accelerator area and energy (roughly 10-20%). Because the two key advantages of hardware accelerators are high throughput and low latency, we filter out a network design point if it introduces more than 10% performance overhead.

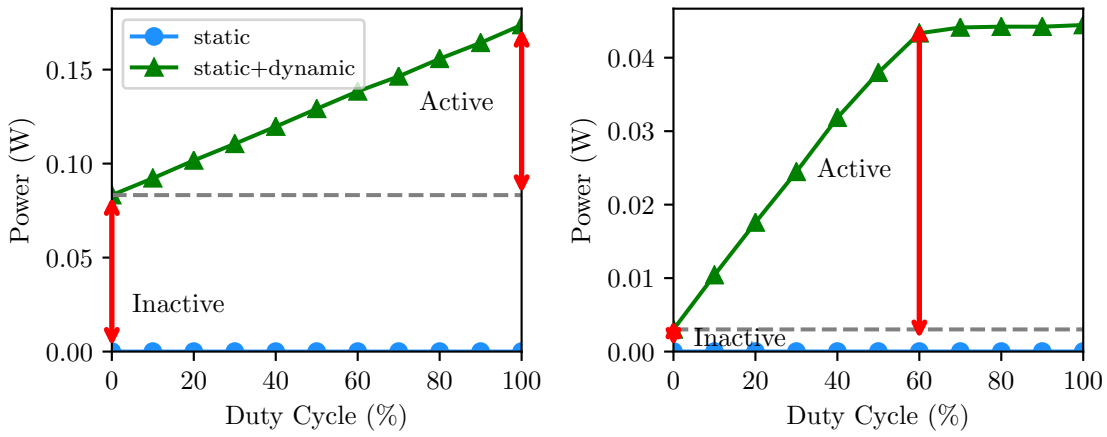


Figure 3.9: Switch and router power with varying duty cycle.

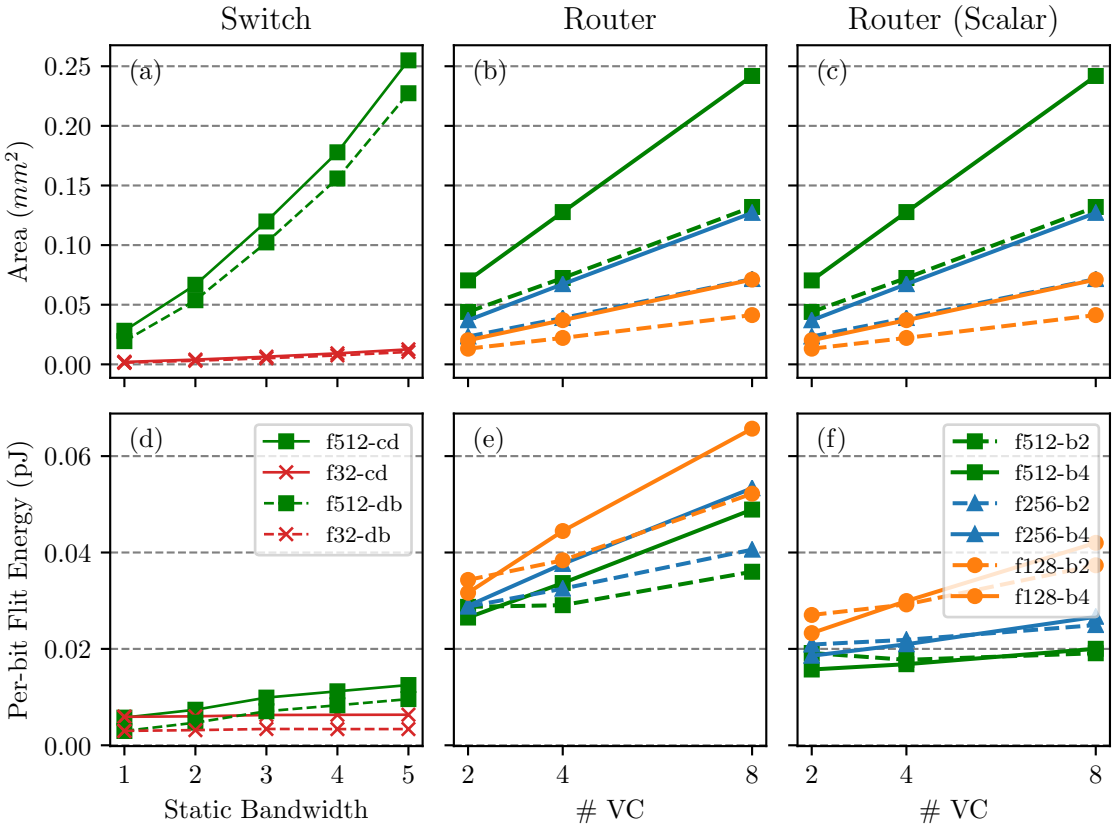
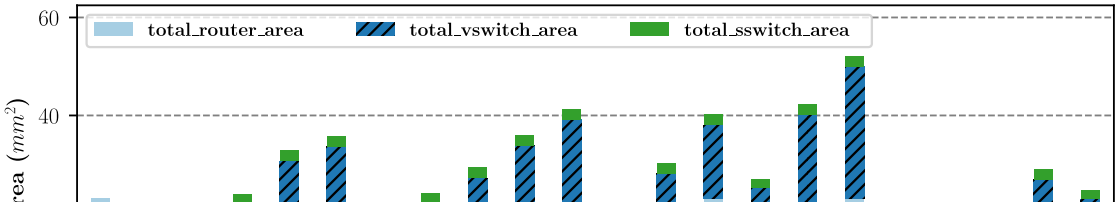


Figure 3.10: Area and per-bit energy for (a,d) switches and (b,c,e,f) routers. (c,f) Subplots (c,f) show area and energy of the vector router when used for scalar values (32-bit).



Notation	Description
[S,H,D]	Static, hybrid, and dynamic network
x#	Static bandwidth on vector network (#links between switches)
f#	Flit width of a router or vector width of a switch
v#	Number of VC in router
b#	Number of buffers per VC in router
[db,cd]	Buffered vs. credit-based flow control in switch

Table 3.2: Network design parameter summary.

This is calculated by comparing to an ideal network with infinite bandwidth and zero latency.

For metrics that are calculated per application, such as performance, performance/watt, and power efficiency, we first normalize the metric with respect to the worst network configuration for that application. For each network configuration, we present a geometric mean normalized across all applications. For all of our experiments, except Section 3.2.4, we use a network size of 14×14 end-point PBs. All vector networks use a vectorization factor of 16 (512 bit messages).

Bandwidth scaling with network size

Figure 3.12 shows how different networks allow several applications to scale to different numbers of PBs. For IO-bound applications (BlackScholes and TPCHQ6), performance does not scale with additional compute and on-chip memory resources. However, the performance of compute-bound applications (GEMM and SGD) improves with increased resources, but plateaus at a level that is determined by on-chip network bandwidth. This creates a trade-off in accelerator design between highly vectorized compute PBs with a small network—which would be underutilized for non-vectorized problems—and smaller compute PBs with limited performance due to network overhead. For more finely grained compute PBs, both more switches and

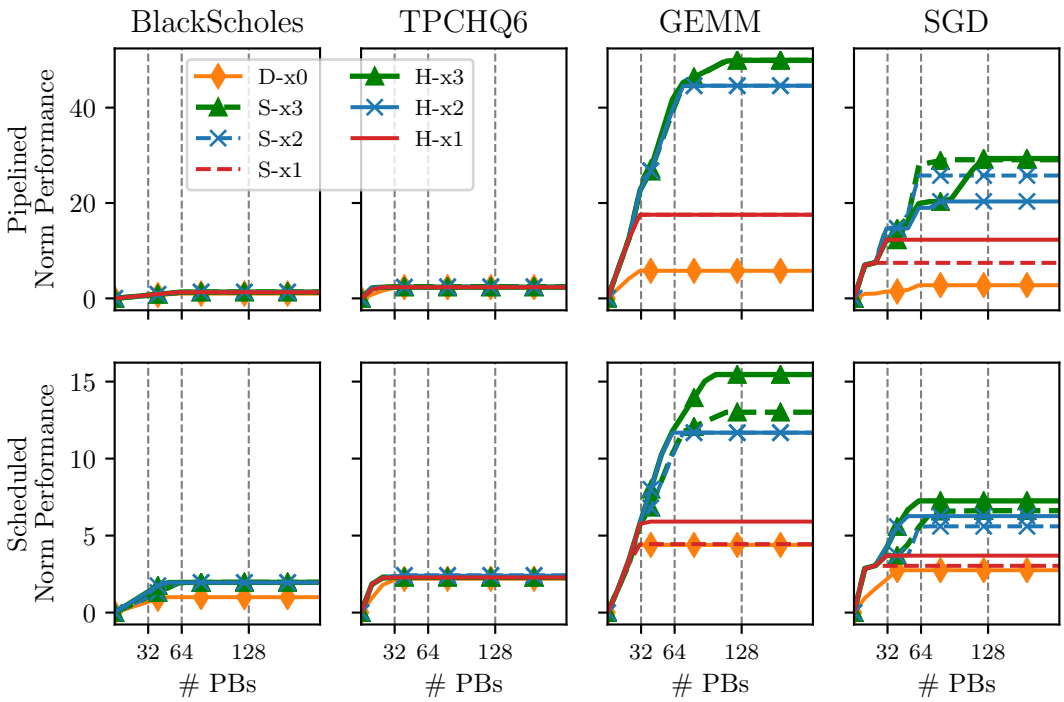


Figure 3.12: Performance scaling with increased CGRA grid size for different networks.

more costly (higher-radix) switches must be employed to meet application requirements.

The scaling of time-scheduled accelerators (bottom row) is much less dramatic than that of deeply pipelined architectures (top row). Although communication between PBs in these architectures is less frequent, the scheduled architecture must use additional parallelization to match the throughput of the pipelined architecture; this translates to larger network sizes.

For pipelined architectures, both hybrid and static networks provide similar scaling with the same static bandwidth: the additional bandwidth from the dynamic network in hybrid networks does not provide additional scaling. This is mostly due to a bandwidth bottleneck between a PB and its router, which prevents the PB from requesting multiple elements per cycle. Hybrid networks tend to provide better scaling for time-scheduled architectures; multiple streams can be time multiplexed at each ejection port without losing performance.

Bandwidth and flow control in switches

In this section, we study the impact of static network bandwidth and flow control mechanism (per-hop vs. end-to-end credit-based). On the left side of Figure 3.14, we show that increased static bandwidth results in a linear performance increase and a superlinear increase in area and power. As shown in Section 3.2.4, any increase in accelerator size must be coupled with increased network bandwidth to effectively scale performance. This indicates that network overhead will increase with the size of an accelerator.

The right side of Figure 3.14 shows that, although credit-based flow control reduces the amount of buffering in switches and decreases network area and energy, application performance is significantly impacted. This is the result of imbalanced data-flow pipelines in the program: when there are parallel long and short paths over the network, there must be sufficient buffer space on the short path equal to the product of throughput and the difference in latency. Because performance is our most important metric, credit-based flow control is not feasible, especially because the impact of bubbles increases with communication distance, and therefore network

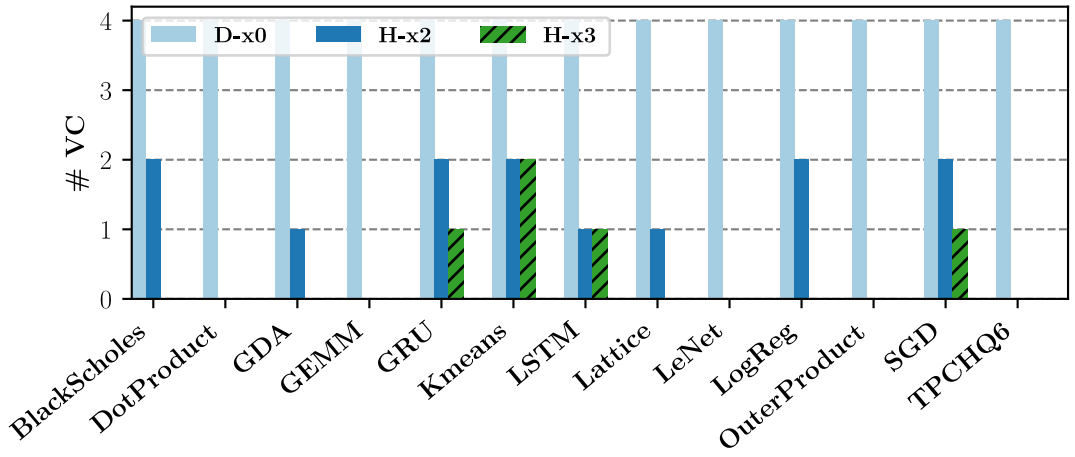


Figure 3.13: Number of VCs required for dynamic and hybrid networks. (No VCs indicates that all traffic is mapped to the static network.)

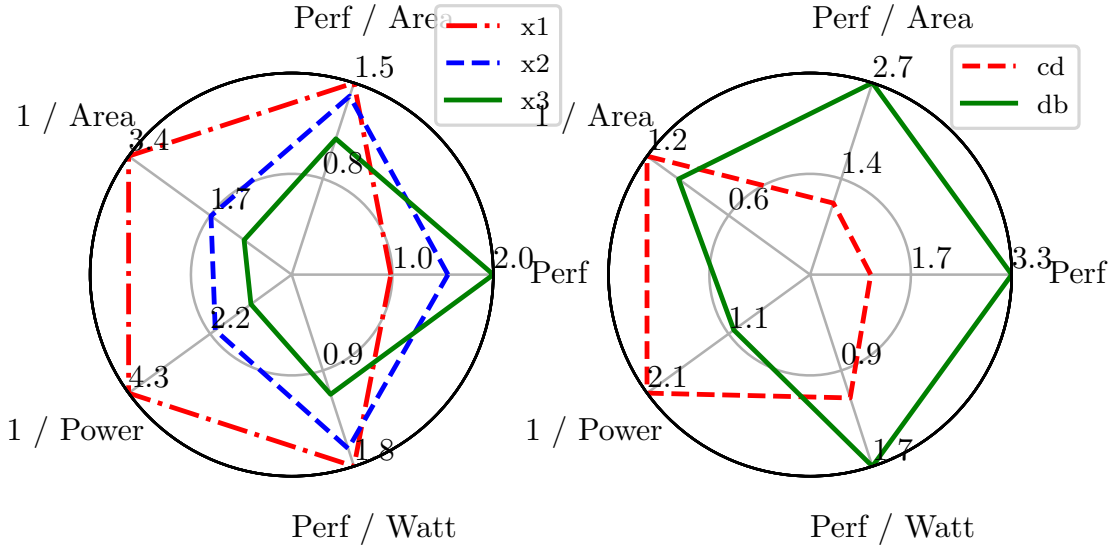
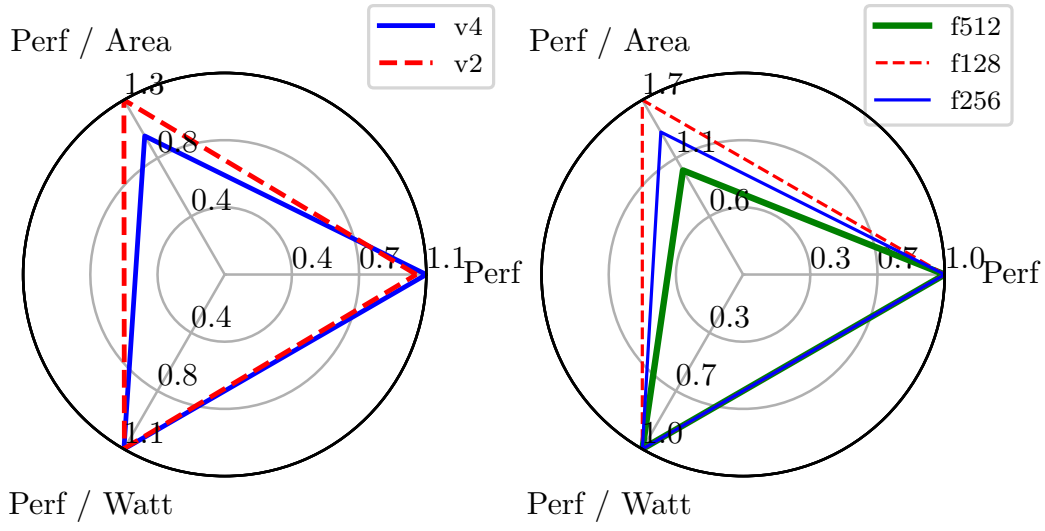


Figure 3.14: Impact of bandwidth and flow control strategies in switches.



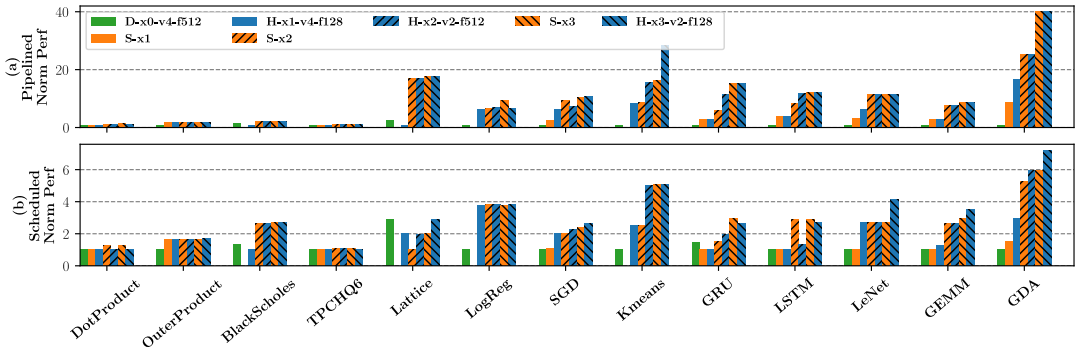


Figure 3.17: Normalized performance for different network configurations.

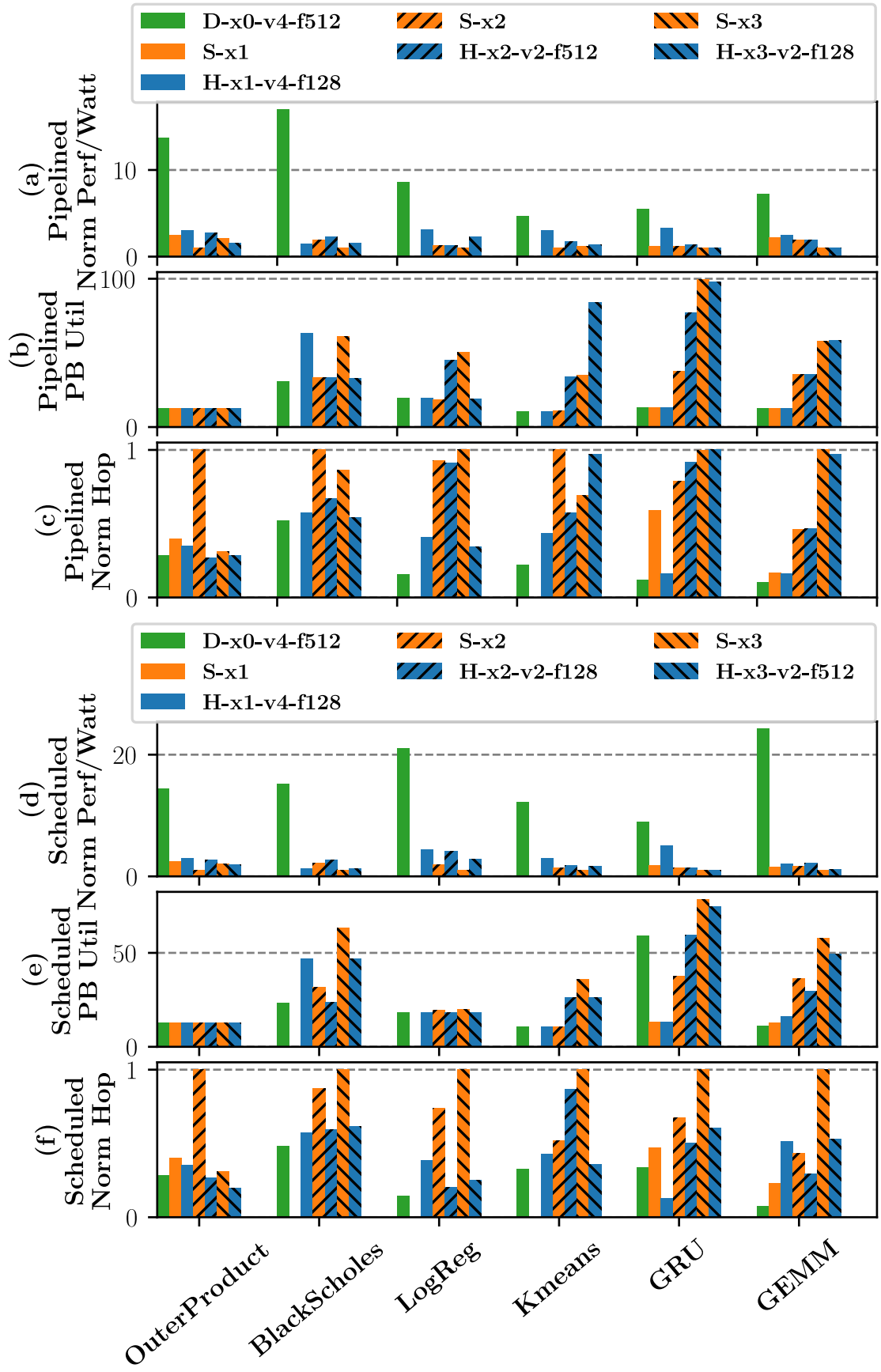
size.

VC count and reduced flit width in routers

In this experiment, we study the area-energy-performance trade-off between routers with different VC counts. As shown in Section 3.2.3, using many VCs increases both network area and energy. However, using too few VCs may force roundabout routing on the dynamic network or result in VC allocation failure when the network is heavily utilized. Nonetheless, the left side of Figure 3.15 shows minimal performance improvement from using more VCs.

Therefore, for each network design, we use a VC count equal to the maximum number of VCs required to map all applications to that network. Figure 3.13 shows that the best hybrid network configurations with 2x and 3x static bandwidth require at most 2 VCs, whereas the pure dynamic network requires 4 VCs to map all applications. Because dynamic network communication is infrequent, hybrid networks with fewer VCs provide both better energy and area efficiency than networks with more VCs, even though this constrains routing on the dynamic network.

We also explore the effects of reducing dynamic network bandwidth by using smaller routers; as shown in Section 3.2.3, routers with smaller flits have a much smaller area. Ideally, we could scale static network bandwidth while using a low-bandwidth router to provide an escape path and reduce overall area and energy overhead. The right side of Figure 3.15 shows that, for a hybrid network, reducing


D-x0-v4-f512
S-x1
H-x1-v4-f128
S-x2
H-x2-v2-f512
S-x3
H-x3-v2-f128
(a) Pipelined Norm Perf/Watt
(b) Pipelined PB Util
(c) Pipelined Norm Hop
D-x0-v4-f512
S-x1
H-x1-v4-f128
S-x2
H-x2-v2-f128
S-x3
H-x3-v2-f512
(d) Scheduled Norm Perf/Watt
(e) Scheduled PB Util
(f) Scheduled Norm Hop
OuterProduct
BlackScholes
LogReg
Kmeans
GRU
GEMM

flit width improves area efficiency with minimal performance loss.

Static vs. hybrid vs. dynamic networks

Figure 3.17 shows the normalized performance for each application running on several network configurations. For some applications, the bar for S-x1 is missing; this indicates that place and route failed for all unrolling factors. For DRAM-bound applications, the performance variation between different networks is trivial because only a small fraction of the network is being used. In a few cases (Kmeans and GDA), hybrid networks provide better performance due to slightly increased bandwidth. For compute-bound applications, performance primarily correlates with network bandwidth because more bandwidth permits a higher parallelization factor.

The highest bandwidth static network uses the most PBs, as shown in Figures 3.18(b,e), because it permits more parallelization. It also has more data movement, as shown in (c,f), because PBs can be distributed farther apart. Due to bandwidth limitations, low-bandwidth networks perform best with small unrolling factors—they are unable to support the bisection bandwidth of larger program graphs. This is evident in Figures 3.18(b,e), where networks D-x0-v4-f512 and S-x2 have small PB utilizations.

With the same static bandwidth, most hybrid networks have better energy efficiency than the corresponding pure static networks, even though routers take more energy than switches to transmit the same amount of data. This is a result of allowing a small amount of traffic to escape onto the dynamic network: with the dynamic network as a safety net, static place and route tends to converge to better placements with less overall communication. This can be seen in Figures 3.18(c,f), where most static networks have larger hop counts than the corresponding hybrid network; hop count is the sum of all runtime link traversals, normalized per-application to the network configuration with the most hops. Subplots (e,f) show that more PBs are utilized with static networks than hybrid networks. This is because the compiler imposes less stringent IO constraints on PBs when partitioning for the hybrid network (as explained in Section **??**), which results in fewer PBs, less data movement, and greater energy efficiency for hybrid networks.

In Figure 3.16, we summarize the best perf/watt and perf/area (among network configurations with ¡10% performance overhead) for pipelined and scheduled CGRA architectures. Pure dynamic networks are not shown because they perform poorly due to insufficient bandwidth. On the pipelined CGRA, the best hybrid network provides a 6.4x performance increase, 2.3x better energy efficiency, and a 6.9x perf/area increase over the worst network configuration. The best static network provides 7x better performance, 1.2x better energy efficiency, and 6.3x better perf/area. The hybrid network gives the best perf/area and perf/watt, with a small degradation in performance when compared to the static network. On the time-scheduled CGRA, both static and hybrid networks have an 8.6x performance improvement. The hybrid network gives a higher perf/watt improvement at 2.2x, whereas the static network gives a higher perf/area improvement at 2.6x. Overall, the hybrid networks deliver better energy efficiency with shorter routing distances by allowing an escape path on the dynamic network.

Chapter 4

Compiler

Chapter 5

Conclusions

18 For I testify unto every man that heareth the words of the prophecy of this book, If any man shall add unto these things, God shall add unto him the plagues that are written in this book:

19 And if any man shall take away from the words of the book of this prophecy, God shall take away his part out of the book of life, and out of the holy city, and from the things which are written in this book.

Appendix A

Appendix

18 For I testify unto every man that heareth the words of the prophecy of this book, If any man shall add unto these things, God shall add unto him the plagues that are written in this book:

19 And if any man shall take away from the words of the book of this prophecy, God shall take away his part out of the book of life, and out of the holy city, and from the things which are written in this book.