STATIC VERSUS SCHEDULED INTERCONNECT IN COARSE-GRAINED RECONFIGURABLE ARRAYS

Brian Van Essen,† *Aaron Wood,*‡ *Allan Carroll,*† *Stephen Friedman,*† *Robin Panda,*‡
Benjamin Ylvisaker,† *Carl Ebeling,*† *and Scott Hauck*‡

†Dept. of Computer Science and Engineering and ‡Dept. of Electrical Engineering
University of Washington, Seattle, WA 98195
†email:{vanessen, allanca, sfriedma, ben8, ebeling}@cs.washington.edu
‡email: {arw82, robin, hauck}@ee.washington.edu

ABSTRACT

Spatially-tiled architectures, such as Coarse-Grained Reconfigurable Arrays (CGRAs), are powerful architectures for accelerating applications in the digital-signal processing, embedded, and scientific computing domains. In contrast to Field-Programmable Gate Arrays (FPGAs), another common accelerator, they typically time-multiplex their processing elements and are word rather than bit-oriented. These differences lead us to re-examine some of the traditional architecture choices made for FPGAs as we move to these coarser-granularity architectures. In this paper we study the efficiency of time-multiplexing global interconnect as architectures scale from single-bit to multi-bit datapaths.

Using the Mosaic infrastructure, we analyzed the design trade-offs involved in static vs. time-multiplexed routing for global interconnect channels, as well as the benefit of including a dedicated bit-wide control interconnect to supplement the word-wide datapath of a CGRA. We show that a time-multiplexed interconnect is beneficial in these coarse-grained systems, reducing the area-energy product to 0.32× the area-energy product of a fully static interconnect. We also show that for our benchmarks, which include single-bit control logic, providing both word and bit-wide interconnect resources further reduces the area-energy product to 0.94× that of an exclusively word-wide interconnect.

1. INTRODUCTION

The continued scaling of transistor densities has made programmable spatial processors, like Field-Programmable Gate Arrays (FPGAs), increasingly attractive for a variety of computationally demanding applications. Due to the high degree of parallelism available in this family of chips, it is possible to achieve high performance and energy efficiency relative to conventional processors.

Though FPGA-based designs achieve impressive results, application-specific integrated circuits (ASICs) still enjoy a wide performance gap in terms of logic density, clock frequency, and energy efficiency [1]. As many have observed, one major inefficiency in FPGAs is that the majority of logic and routing resources are configured at the bit granularity, even though many applications naturally represent data in 8-, 16-, or 32-bit words. This observation has led to the design and study of a wide variety of coarse-grained reconfigurable arrays (CGRAs), including RaPiD [2], ADRES [3], MATRIX [4], Tartan [5], MorphoSys [6], and HSRA [7].

Coarse-Grained Reconfigurable Arrays are composed of a sea of word-wide processing elements (PEs), distributed storage, and interconnect resources. They are similar to FPGAs, but with ALUs as the fundamental compute element instead of LUTs. Another important difference is that most CGRAs allow the configuration of compute (and possibly interconnect) resources to change on a cycle-by-cycle basis. Supporting this time-multiplexing requires small, distributed configuration memories and some control circuitry, which are amortized across the word-wide resources. Time-multiplexing in this fashion is less attractive for FPGAs because each programming bit in an FPGA controls less logic than in a CGRA. Thus, the additional configuration memory would consume a large portion of the chip.

This paper addresses the question of how flexible the interconnect should be in CGRAs. We focus on the interconnect because it accounts for a large portion of both the area and energy consumption in spatial architectures. At one extreme, FPGA-like architectures require that interconnect resources be configured in exactly one way for the entire run of an application. At the other extreme are architectures that allow every interconnect resource to be configured differently on a cycle-by-cycle basis. In this paper we investigate this tradeoff, comparing architectures at both extremes and mixtures that combine each of these styles of resources.

The optimal tradeoff between scheduled and static resources will likely depend on the word-width of the interconnect, since the overheads associated with some techniques

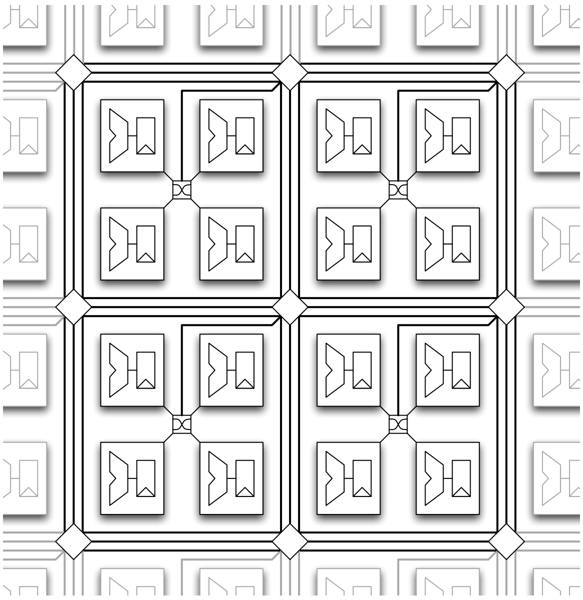

Fig. 1. CGRA Block Diagram - Clusters of 4 PEs connected via a grid of switchboxes.

may be much larger in a 1-bit interconnect than with a 32-bit interconnect. To explore this possibility we consider the area, power, and channel width at different word-widths and maximum hardware supported initiation interval (II), as well as investigating hybrid bitwidth architectures that have a mixture of single-bit and multi-bit interconnect resources.

1.1. Architecture of a CGRA

Like most spatial architectures, CGRAs are tiled and can be hierarchically clustered. For our studies, compute and storage elements are grouped into processing elements (PEs), which are then interconnected to form clusters. The clusters use a global interconnect to communicate. Figure 1 shows a simple picture of this hierarchical structure using a grid interconnect. Spatial arrangements for PEs and clusters are most commonly meshes or grids [3, 5, 6], which we explore; other patterns, such as linear arrays [2] and fat-pyramids [7], have also been studied.

1.2. Related Work

Recent FPGA architectures include coarse-grained components, such as multipliers, DSP blocks and embedded memories. This trend blurs the line between FPGAs and CGRAs, but even the most extreme commercial architectures still devote most logic resources to single bit components, and to our knowledge no commercial FPGA has word-wide interconnect or a scheduled (time-multiplexed) configuration system.

The interconnects of some early CGRA architectures, such as RaPiD [2] and Matrix [4], combined the concepts of static configuration and scheduled switching to some extent. However, in neither project was there a systematic study of the optimal mixture of the two kinds of resources.

The ADRES [3] and DRESC [8] projects are very similar to the Mosaic project. They provide architecture exploration and CGRA-specific CAD algorithms, but ADRES studies were limited to time-multiplexed interconnect in much smaller CGRA fabrics. In [9], the area and energy tradeoffs between various interconnect patterns are studied. The experiments presented here instead focus on the balance of scheduled and static resources, rather than topology.

2. THE MOSAIC INFRASTRUCTURE

The goal of the Mosaic project [10] is to explore architecture, compiler, and programming language challenges related to CGRAs. To support these explorations, we are creating tools for developing CGRA applications, architectures, compilers, and measurement frameworks. The system is composed of the Macah language and compiler [11], the SPR CGRA mapping tool [12], an architecture generator plugin for Electric VLSI [13], and a Verilog-based simulation system using post-layout SPICE simulation timing and energy data.

2.1. Benchmarks

Table 1 lists our benchmark set. We have carefully constructed inner loops in Macah so that each application has an interesting kernel with sufficient parallelism, which was then mapped through the Mosaic tool chain. The inner loops are software pipelined [15], and the Min II column indicates the minimum number of clock cycles between starting consecutive loop iterations (*i.e.* initiation interval), which is determined by the largest loop carried dependence. The other parameters provide insight into the size and composition of each benchmark, *e.g.* LiveIns is the number of scalar constants loaded into the kernel.

The applications in our interconnect study represent the computationally intensive cores of various algorithms used in DSP, embedded, and scientific computing. We include a FIR filter, 2D convolution, dense matrix multiplication, K-means clustering, a matched filter, CORDIC, and a heuristic motion estimator.

2.2. Compilation

To rapidly explore a variety of architectures, we programmatically compose Verilog primitives to fully specify an architecture instance. The resulting composition is denoted a datapath graph and contains all information needed to perform placement, routing, and power-aware simulation.

Table 1. Benchmark Applications and Simulated Architectures

Application	Min II	# DFG operations ALU	Memory Accesses	Stream IO	LiveIns	# PE clusters	CGRA Grid Size
64-tap FIR filter	2	199	0	1	195	64	10x10
240-tap FIR filter (Banked Mem)	6	284	48	1	177	30	7x8
2D convolution	4	180	6	1	189	30	7x8
8x8 Matrix multiplication	4	361	0	64	218	36	8x8
8x8 Matrix mult. (Small Stream)	9	402	0	16	224	16	6x6
K-means clustering (K=32)	7	525	32	33	189	25	7x7
Matched filter	4	193	30	13	88	20	6x7
Smith-Waterman	5	342	11	5	109	25	7x7
CORDIC	2	178	0	5	46	30	7x8
8×8 Motion estimation [14]	5	465	16	9	189	30	7x8

The compiled Macah program for our infrastructure is a dataflow graph representing the computation that will be mapped to a particular hardware instance. Mosaic maps the dataflow graph onto the CGRA's datapath graph with the SPR CGRA mapping tool [12]. SPR is able to handle a mixture of time-multiplexed and statically configured resources, which allow us to evaluate our different architectural parameters using the same benchmarks. This resulting CGRA configuration is then executed in a Verilog simulator.

2.3. Circuit Area and Energy Modeling

Recognizing that the energy consumption of a logic structure can be highly dependant on the input data sequence [16], we use simulation-driven power modeling. We characterized the fundamental interconnect circuits from full-custom layouts in a 65nm process. With this approach, we provide realistic values for a particular process and, more importantly, comparable results for different topologies, allowing us to assess architectural decisions.

We use the Verilog PLI to account for energy consumption as the simulation progresses. The values accumulated are derived from our transistor-level circuit models. The high level of detail for these simulations requires long runtimes. Currently, simulating a large highly utilized architecture for a few tens of thousand clock cycles can take as long as 20 hours. To reduce this burden we scale the input sets down to small, but non-trivial, sizes. Applications that are well-suited to CGRAs tend to be regular enough that the behavior of a kernel on a small set of inputs is representative of the behavior on much larger set of inputs.

3. EXPERIMENTAL SETUP

The channel width of our CGRA, like an FPGA, is the number of independent communication channel pairs between clusters of PEs. Thus, on a 32-bit architecture one channel (or "track") contains 32 wires, and a channel pair has two unidirectional tracks, oriented in opposite directions. A primary goal of this study is to understand the tradeoff between required minimum channel width and the flexibility of each channel, *i.e.* is it statically configured or scheduled. A statically configured channel is like an FPGA interconnect track; it is configured once for a given application. A scheduled channel changes its communication pattern from cycle to cycle of execution, iterating through its schedule of configurations. In our architectures the scheduled channels use a modulo-counter and a cyclic-schedule.

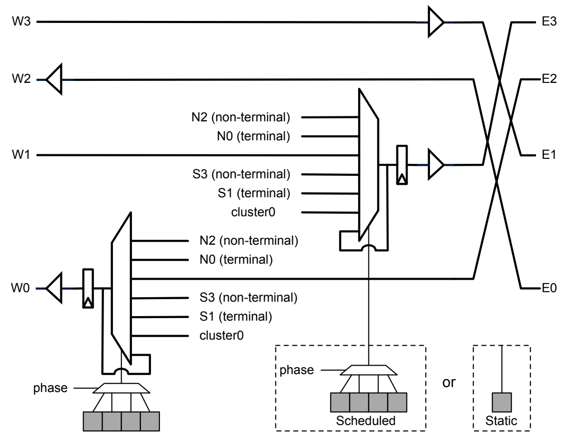

Fig. 2. Block diagram of a horizontal slice of a switchbox with channel span 2. Shaded grey boxes represent configuration SRAM.

To explore the benefits versus overhead of these scheduled interconnect channels, we mapped and simulated the benchmarks described in Table 1 to a family of CGRAs. These architectures vary in the ratio of scheduled to statically configured interconnect channels.

3.1. Architecture Setup

For this study we explore clustered architectures with a grid topology. Within each CGRA family there was a common set of compute (or PE), IO, and control clusters. The clusters were arranged in a grid, with PE clusters in the center and IO and control clusters on the outer edge . Computing and interconnect resources within the cluster were scheduled, and all scheduled resources supported a maximum II of 16, unless otherwise stated.

The datapath of the PE compute cluster contains 4 arithmetic and logic units (ALUs), 2 memories, 2 sets of loadable registers for live-in scalar values, and additional distributed registers. The control path contains 2 look-up tables (LUTs), 2 flipflops for boolean live-in values, and additional distributed flipflops. Given our focus on the interconnect, the PE cluster is simplified by assuming that the ALU is capable of all C-based arithmetic and logic operations required, including multiplication and data dependent multiplexing. The components in the cluster are connected to each other and a global interconnect switchbox via a scheduled crossbar. Refining the intra-cluster logic and interconnect resources, will be the subject of future work.

The IO clusters are designed to feed data to and from the computing fabric. Each IO cluster contains 4 stream-in and 4 stream-out ports, one set of loadable registers, a set of word-wide and single-bit scan-able registers than can transfer live-out variables to a host processor, and distributed registers for retiming. As with the PE cluster, all components are connected via a scheduled crossbar that also provides access to the associated switchbox. Four of the IO clusters are replaced with control clusters that are the superset of a PE + IO cluster with auxiliary logic that provides loop control.

Each cluster is connected to a switchbox, which are themselves connected to form the global cluster-to-cluster grid. Communication between clusters and switchboxes, and between switchboxes, use single driver, unidirectional channel pairs [17]. The switchbox topology follows a trackgraph-style pattern. All terminal (*i.e.* sourced) tracks are registered leaving the switchbox. Additionally, connections that go into the cluster, or between horizontal and vertical channels, are registered leaving the switchbox. All interconnect channels had a span of 2 – they were registered in every other switchbox. The switchbox layout is shown in Figure 2, based on Figure 5 from [17]. The statically configured channels replace the configuration SRAM bits shown in the bottom of Figure 2 with a single SRAM bit.

In order to determine the length of wires in the global interconnect, we use a conservative estimate of $0.5mm^2$ per cluster, based loosely on a CU-RU pair in the Ambric architecture [18]. The area consumed by the switchbox is computed from our circuit models.

3.2. Benchmark Sizing

Our benchmarks were designed with parameters to adjust the degree of parallelism, the amount of local buffering, etc. Each parameter was then adjusted to a reasonable value that balanced application size versus time to place and route the design and time to simulate the application on an architecture. The circuit was mapped to the smallest near-square (NxN or NxN-1) architecture such that no more than 80% of the ALUs, memories, streams and live-ins were used given the min II of the application. The resulting compilation and simulation times for applications were 30 minutes to 20 hours, depending on the complexity of the application and the size of the architecture.

3.3. VLSI Circuit Modeling and Simulation

Our VLSI layouts of components used to build the interconnect topology provide realistic characterization data for our study. Each component provides area, delay, load, slew rate, inertial delay, static power, and dynamic energy for modeling purposes. Area and static power estimates are based on a sum of the values for the individual components required. The delay for the longest path between registers, is used to estimate a clock period for the device. Registers that are unused in the interconnect are clock-gated to mitigate their load on the clock tree; statically for single-bit channels and word-wide static channels, and on a cycle-by-cycle basis for word-wide scheduled channels.

For simulation of the 8-, 16- and 24-bit interconnect, the architecture was regenerated with these settings. However, our ALU models remained 32-bit and we did not rewrite the benchmarks for correct functionality with 8-, 16-, or 24-bit widths. Instead, we only recorded transitions on the appropriate number of bits, masking off the energy of the upper bits in the interconnect. This approximated the interconnect power for applications written for narrower bit widths.

3.4. Static versus Scheduled Channels

Scheduled interconnect channels can be reconfigured on a cycle-by-cycle basis, which carries a substantial cost in terms of configuration storage and management circuitry. Static configuration clearly offers less flexibility, but at a lower hardware cost. We explore the effects of different mixtures of scheduled and static channels by setting the scheduled-to-static ratio and then finding the minimum channel width needed to map a benchmark. For example, with a 70%/30% scheduled-to-static split, if the smallest channel width K-means mapped to was 10 channels, 7 would be scheduled and 3 static.

We measure the area needed to implement static and scheduled channels, and the energy dissipated by those circuits during the execution of each benchmark. At a given channel width, lower scheduled-to-static ratios are better area-wise. However, as we decrease the scheduled-to-static ratio, we expect that the number of tracks required will increase, thus increasing the area and energy costs. Note that our placement and routing tools [12] support static-sharing, which attempts to map multiple signals onto static tracks in

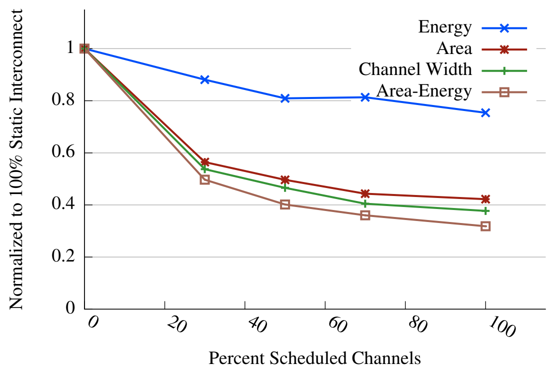

Fig. 3. Area, channel width, energy, and area-energy metrics as 32-bit interconnect becomes more scheduled.

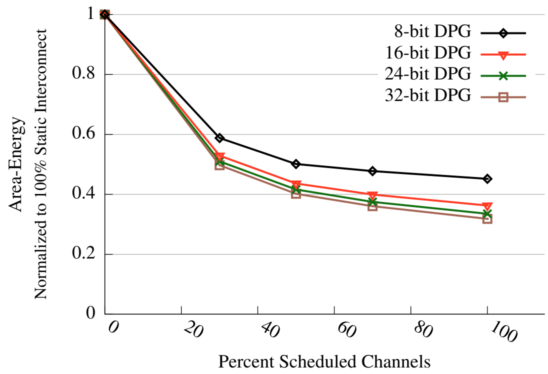

Fig. 4. Area-energy product for different datapath word-widths, as the interconnect becomes more scheduled.

different phases of the schedule; the signals have to configure the track in the same way so that a single configuration works for all signals that share a track.

4. RESULTS AND ANALYSIS

We divide our results into four sections, each focusing on a different issue in the interconnect. We start with a 32-bit interconnect and adjust the scheduled-to-static channel ratio. Next we vary the interconnect width down to 8-bit. From there we look at the impact of maximum II supported by the hardware. The last section explores the addition of single-bit interconnect resources for control signals.

To generate the data, we ran each benchmark with three random placer seeds and used the result that gave the best throughput (lowest achieved II), followed by smallest area and lowest energy consumption. By using placements with the same II across the entire sweep, the performance is independent of the scheduled-to-static channel ratio. The data was then averaged across all benchmarks.

4.1. Interconnect Scheduled/Static Ratio

Intuitively, varying the ratio of scheduled-to-static channels is a resource balancing issue of fewer “complex” channels versus more “simple” channels. Depending on the behavior of the applications, we expect that there will be a sweet spot for the scheduled-to-static ratio. Figure 3 summarizes the results for channel width, area, energy, and area-energy product when varying the ratio of scheduled-to-static channels in the interconnect. Each point is the average across our benchmark suite for the given percentage of scheduled channels. The data is for 32-bit interconnects and hardware support for II up to 16. All results are normalized to the fully static (0% scheduled) case. We observe improvements in all metrics as the interconnect moves away from static configuration, all the way to 100% scheduled. By making the interconnect more flexible, the number of required channels is reduced to 0.38×, which translates into 0.42× area and 0.75× energy consumption. This is despite the area and energy overhead required to provide the configurations to these scheduled channels. Overall, the area-energy product is reduced to 0.32× that of the fully static interconnect.

4.2. Datapath Word-Width and Scheduled/Static Ratio

For a 32-bit interconnect, Figure 3 shows that having entirely scheduled channels is beneficial. However, for lower bitwidths, the overhead of scheduled channels is a greater factor. Figure 4 shows the area-energy trends for architectures with different word-widths. We observe that fully scheduled is best for all measured bitwidths, but as the datapath narrows from 32-bits down to 8-bits the advantage of a fully scheduled interconnect is reduced. However, it is still a dramatic improvement over the fully static baseline. Our previous result for a 32-bit interconnect showed a 0.32× reduction in area-energy product. With the narrower interconnect, we see reductions to 0.34×, 0.36×, and 0.45× the area-energy product of the fully static baseline for 24-, 16- and 8-bit architectures respectively.

To explain this trend, we first look at the energy consumed when executing a benchmark in more detail, followed by the area breakdown of the interconnect. Figure 5 shows the energy vs scheduled-to-static ratio in the categories:

- **signal** - driving data through a device or along a wire
- **cfg** - reconfiguring dynamic multiplexors and static and dynamic energy in the configuration SRAM
- **clk** - clocking configured registers
- **static** - static leakage (excluding configuration SRAM)

There are two interesting behaviors to note: first, the static channels have less energy overhead in the configuration system. So, if two CGRAs have the same number of channels then the energy consumed will go down as the percentage of static channels increased. However, the overhead

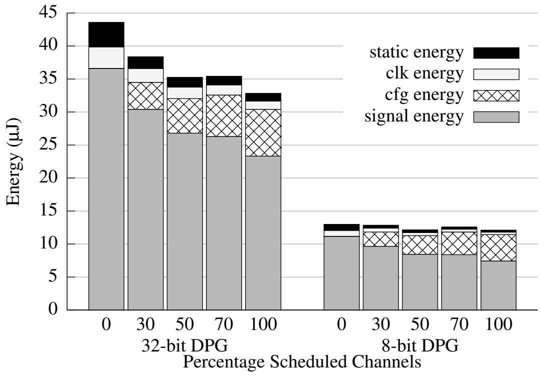

Fig. 5. Average energy for global routing resources.

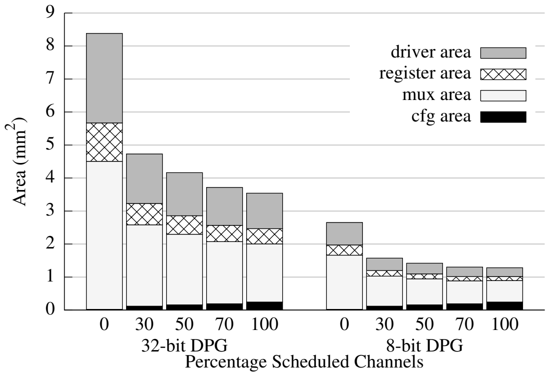

Fig. 6. Average area for global routing resources.

of a scheduled channel is small at a datapath width of 32-bits, and so the energy reduction is small. The percentage of the energy budget consumed by configuration overhead does increase dramatically at narrower datapath widths.

The second behavior is a non-obvious side-effect of sharing static channels, as detailed in [12]. When multiple signals share a portion of a static interconnect they form a large fanout tree that is active in each phase of execution. As a result, each individual signal is driving a larger load than necessary, and so the additional dynamic energy consumed outweighs the energy saved due to reduced overhead. This behavior is one of the contributing factors that forces the signal energy to go up as the interconnect becomes more static.

Figure 6 details the breakdown of the interconnect area vs. the scheduled-to-static ratio. As with energy overhead, we observe that the configuration system's area is tiny for fully static systems, and a small portion of the fully scheduled 32-bit interconnect. However, at narrower datapath widths, the area overhead of the configuration system accounts for a non-trivial portion of the interconnect, up to 19% for the fully scheduled case. All other areas are directly dependent on the channel width of the architecture, and dominate the configuration area. As channels become more scheduled, the added flexibility makes each channel more useful, reducing the number of channels required.

In summary, Figures 4, 5, and 6 show that the overhead required to implement a fully scheduled interconnect is reasonably small. However, as the bitwidth of the datapath narrows, those overheads become more significant. At bitwidths below 8-bit a static interconnect will likely be the most energy-efficient, though a scheduled interconnect is likely to be the most area-efficient until we get very close to a single-bit interconnect.

4.3. Hardware Supported Initiation Interval

Each architecture has a maximum initiation interval, set by the number of configurations for the CGRA that can be stored in the configuration SRAM. For an application to map to the CGRA the application's II must be less that or equal to the CGRA's maximum supported II.

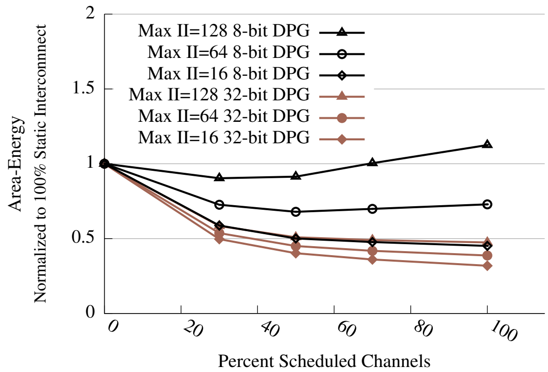

Fig. 7. Area-energy product for 32- and 8-bit datapath word-widths and maximum supported II of 64 and 128 configurations, as the interconnect becomes more scheduled.

Our results indicate that scheduled channels are beneficial for word-wide interconnect. However, the overhead associated with scheduled channels changes with the number of configurations supported. Figure 7 shows the area-energy curves for 32-bit and 8-bit datapaths with hardware that supports an II of 16, 64 or 128. The curve for an II of 128 on an 8-bit datapath is most interesting. In this configuration the overhead of the scheduled channels dominates any flexibility benefit. For the other cases, fully scheduled is a promising answer. Note that support for a max II of 16 to 64 in the hardware is a reasonable range in the design space, given that 9 is the largest II requirement for any of our existing benchmarks. A related study [19] showed that an II of 64 would cover >90% of loops in MediaBench and SPECFP, and that the maximum II observed was 80.

4.4. Augmenting CGRAs with Control Specific Resources

The datapath applications we used for benchmarks are dominated by word-wide operation; however, there is still a non-

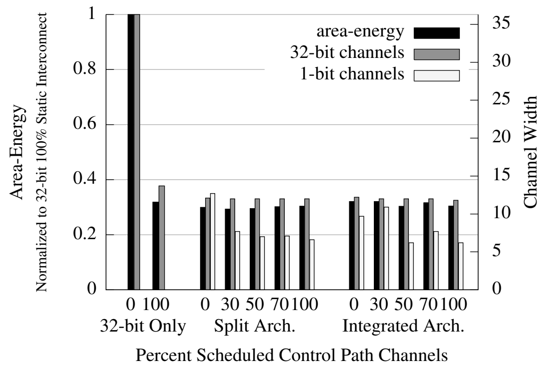

Fig. 8. Comparison of 32-bit only, split, and integrated interconnects. Area-energy is shown on the left y-axis and 32- and 1-bit channel width on the right y-axis.

trivial amount of single-bit control logic within each kernel. One major source of this control logic is due to if-conversion and predication; another is loop bound control. Since our architectures can only execute a fixed schedule, to achieve data dependent control flow `if`-statements must be predicated such that both possible outcomes are executed and the correct result is selected using the single-bit predicate. It is common in our applications for 20-30% of the dataflow graph to be control logic. Routing 1-bit signals in a 32-bit interconnect wastes 31 wires in each channel. Given the observations of §4.2, the optimal interconnect will be different for control and datapath signals.

The obvious alternative is an interconnect that includes both 32-bit and 1-bit resources, optimized to the demands of each signal type. We experiment with two distinct combinations of word-wide and single-bit interconnects, split and integrated. For the split architecture, there are disjoint single- and multi-bit interconnects, where each port on a device is connected to the appropriate interconnect. For the integrated architecture datapath signals are restricted to the 32-bit interconnect, while control signals can use either the 32-bit or 1-bit interconnect, hopefully reducing inefficiencies from resource fragmentation. The integrated and split architecture use a fully scheduled 32-bit interconnect and the 1-bit interconnect is tested with a range of scheduled-to-static ratios.

As we see in Figure 8, adding a single-bit interconnect reduces the channel width of the multi-bit interconnect. This translates into a reduced area-energy product, though it did not vary significantly with the single-bit interconnect's scheduled-to-static ratio. Given a similar area-energy product, a fully static single-bit interconnect is simpler to design and implement, and thus may be preferable. The variations in the single-bit channel width for the integrated architecture likely results from the increased complexity in the routing search space, making it more challenging for the heuristic algorithms to find good solutions. Even if we only examine the benchmarks that achieved a smaller channel width in the integrated tests, we see that there is no measurable improvement in area-energy product. This leads us to conclude that the integrated architecture is not worth the additional complexity or reduction in SPR's performance.

We can compare the results for a split architecture with a scheduled 32-bit and a static 1-bit interconnect to the scheduled 32-bit only baseline from 4.1. We see a reduction in the number of scheduled channels to 0.88× that of the baseline, though there should be roughly one 1-bit channel for each 32-bit channel. Alternatively, we can view this as replacing 1.6 32-bit channels with 12.7 1-bit channels. Note that the routing inside the clusters is also more efficient in the split architecture than the 32-bit only architecture, since separate control and data crossbars will be more efficient than one large integrated crossbar. This effect is beyond the scope of our current measurement setup. Comparing split to scheduled 32-bit only, the overall area and energy are reduced to 0.98× and 0.96× respectively, and area-energy product improves to 0.94×. The split architecture's area-energy product is 0.30× that of the 100% static datapaths without dedicated control resources.

5. CONCLUSIONS

In this paper we explored the benefits of time-multiplexing the global interconnect for CGRAs. We found that for a word-wide interconnect, going from 100% statically configured to 100% scheduled (time-multiplexed) channels reduced the channel width to 0.38× the baseline. This in turn reduced the the energy to 0.75×, the area to 0.42×, and the area-energy product to 0.32×, despite the additional configuration overhead. This is primarily due to amortizing the overhead of a scheduled channel across a multi-bit signal. It is important to note that as the datapath width is reduced, approaching the single bit granularity of an FPGA, the scheduled channel overhead becomes more costly. We find that for datapath widths of 24-, 16-, and 8-bit, converting from fully static to fully scheduled reduces area-energy product to 0.34×, 0.36×, and 0.45×, respectively.

Another factor that significantly affects the best ratio of scheduled versus static channels is the maximum degree of time-multiplexing supported by the hardware, *i.e.* its maximum II. Supporting larger II translates into more area and energy overhead for scheduled channels. We show that for a 32-bit datapath, supporting an II of 128 is only 1.49× more expensive in area-energy than an II of 16, and a fully scheduled interconnect is still a good choice. However, for an 8-bit datapath and a maximum II of 128, 70% static (30% scheduled) achieves the best area-energy performance, and fully static is better than fully scheduled.

Lastly, while CGRAs are intended for word-wide applications, the interconnect can be further optimized by providing dedicated resources for single-bit control signals. In general, we find that augmenting a fully scheduled datapath interconnect with a separate, fully static, control-path interconnect reduces the number of datapath channels to $0.88\times$ and requires a control-path of roughly equal size. As a result the area-energy product is reduced to $0.94\times$. This leads to a total area-energy of $0.30\times$ the base case of a fully static 32-bit only interconnect, a $3.3\times$ improvement.

In summary, across our DSP and scientific computing benchmarks, we have found that a scheduled interconnect significantly improves channel width, area, and energy of systems with moderate to high word-widths that support a reasonable range of time-multiplexing. Furthermore, the addition of a single-bit, fully static, control-path is an effective method for offloading control and predicate signals, thus further increasing the efficiency of the interconnect.

6. ACKNOWLEDGEMENTS

This work was supported by Department of Energy grant #DE-FG52-06NA27507, and NSF grants #CCF0426147 and #CCF0702621. Allan Carroll was supported by an NSEDG Fellowship.

7. REFERENCES

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: ACM Press, 2006, pp. 21–30.

[2] C. Ebeling, D. C. Cronquist, and P. Franklin, "RaPiD - Reconfigurable Pipelined Datapath," in *International Workshop on Field-Programmable Logic and Applications*, R. W. Hartenstein and M. Glesner, Eds. Springer-Verlag, Berlin, 1996, pp. 126–135.

[3] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *International Conference on Field-Programmable Logic and Applications*, vol. 2778, Lisbon, Portugal, 2003, pp. 61–70, 2003.

[4] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996, pp. 157–166.

[5] M. Mishra and S. C. Goldstein, "Virtualization on the Tartan Reconfigurable Architecture," in *International Conference on Field-Programmable Logic and Applications*, 2007, pp. 323–330.

[6] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[7] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon, "HSRA: high-speed, hierarchical synchronous reconfigurable array," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM Press, 1999, pp. 125–134.

[8] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "DRESC: a retargetable compiler for coarse-grained reconfigurable architectures," in *IEEE International Conference on Field-Programmable Technology*, 2002, pp. 166–173.

[9] A. Lambrechts, P. Raghavan, M. Jayapala, B. Mei, F. Catthoor, and D. Verkest, "Interconnect Exploration for Energy Versus Performance Tradeoffs for Coarse Grained Reconfigurable Architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 1, pp. 151–155, Jan. 2009.

[10] "Mosaic Research Group." [Online]. Available: http://www.cs.washington.edu/research/lis/mosaic/

[11] A. Carroll, S. Friedman, B. Van Essen, A. Wood, B. Ylvisaker, C. Ebeling, and S. Hauck, "Designing a Coarse-grained Reconfigurable Architecture for Power Efficiency," Department of Energy NA-22 University Information Technical Interchange Review Meeting, Tech. Rep., 2007.

[12] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, "SPR: an architecture-adaptive CGRA mapping tool," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: ACM, 2009, pp. 191–200.

[13] Sun Microsystems and Static Free Software, "Electric VLSI Design System." [Online]. Available: http://www.staticfreesoft.com/

[14] C. Zhu, X. Lin, L. Chau, and L.-M. Po, "Enhanced Hexagonal Search for Fast Block Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 10, pp. 1210–1214, October 2004.

[15] M. Lam, "Software pipelining: an effective scheduling technique for VLIW machines," in *ACM SIGPLAN conference on Programming Language design and Implementation*. New York, NY, USA: ACM Press, 1988, pp. 318–328.

[16] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[17] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *IEEE International Conference on Field-Programmable Technology*, Dec. 2004, pp. 41–48.

[18] M. Butts, A. M. Jones, and P. Wasson, "A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing," in *IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 55–64.

[19] N. Clark, A. Hormati, and S. Mahlke, "VEAL: Virtualized Execution Accelerator for Loops," in *IEEE International Symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 389–400.