

Managing short-lived and long-lived values in Coarse-Grained Reconfigurable Arrays

Brian Van Essen*, Robin Panda†, Aaron Wood†, Carl Ebeling*, Scott Hauck†

*Dept. of Computer Science and Engineering and †Dept. of Electrical Engineering

University of Washington, Seattle, WA 98195

Email: *{vanessen, ebeling}@cs.washington.edu and †{robin, arw82, hauck}@ee.washington.edu

Abstract—Efficient storage in spatial processors is increasingly important as such devices get larger and support more concurrent operations. Unlike sequential processors that rely heavily on centralized storage, *e.g.* register files and embedded memories, spatial processors require many small storage structures to efficiently manage values that are distributed throughout the processor’s fabric. The goal of this work is to determine the advantages and disadvantages of different architectural structures for storing values on-chip when optimizing for energy efficiency as well as area.

Examination of applications for coarse-grained reconfigurable arrays (CGRAs) shows that most values are short-lived; they are produced and consumed quickly, but the distribution of value lifetimes has a reasonably long tail. We take advantage of this distribution to optimize register storage structures for managing short-, medium-, and long-lived values.

We show that using a combination of register storage structures, each tailored for values with different lifetimes, provides a reduction in overall area-energy product to $0.69\times$ the area-energy of the baseline architecture, without loss of performance. Finally we provide energy profiles, characteristics, and comparisons of each register structure to enable architects to guide future design choices.

Keywords—CGRA; energy-efficiency; storage; architecture;

I. INTRODUCTION

Spatial processors, such as field programmable gate arrays (FPGAs), massively-parallel processor arrays (MPPAs), and coarse-grained reconfigurable arrays (CGRAs), accelerate applications by distributing operations across many parallel compute resources. As a result, these spatial architectures are unable to efficiently take advantage of very large, centralized storage structures that are commonly found in sequential processors. To maintain a high level of performance, it is necessary to have storage structures distributed throughout the architecture that hold all live values and provide the required bandwidth.

There are several possible design choices for these small distributed storage structures, but to date there hasn’t been a concise comparison of their area and

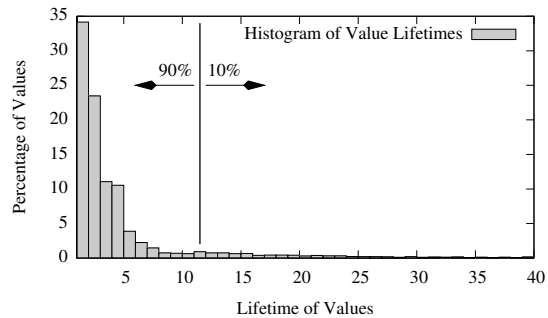


Figure 1. Histogram of value lifetimes for Mosaic benchmarks.

energy trade-offs. In this paper we explore the role of specialized register structures for managing short-lived and long-lived values in coarse-grained reconfigurable arrays. Furthermore, we characterize the area and energy costs for several register structures and quantify their advantages and disadvantages.

II. CHALLENGES

A common use case for CGRAs is accelerating the inner loops of an application. One technique for exploiting loop-level parallelism is to software pipeline the loop, which is then executed using a modulo schedule. Modulo scheduled loops are a very efficient technique for executing loops on CGRAs. Architectures such as ADRES [1], MorphoSys [2], and MATRIX [3], use (or support) a statically computed modulo schedule for controlling ALU, storage, and routing resources.

Modulo schedules [4], [5] were developed for VLIW processors to provide a compact encoding of the loop while maintaining high performance. A modulo schedule is executed one or more times for each iteration of the application’s loop, and each pass through the modulo schedule is referred to as a wave. The rate at which the schedule restarts is the initiation interval (II), and is inversely proportional to the loop’s throughput.

We characterize the lifetimes for values as short,

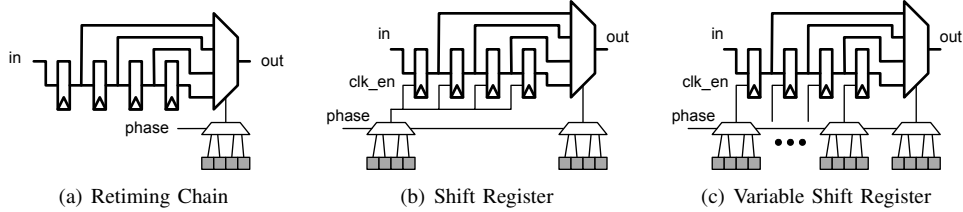


Figure 2. Retiming chain and shift register structures containing 4 registers.

medium, and long. Short-lived values have only one or a few cycles between being produced and consumed, which is typically less than the loop’s Π . An intermediate lifetime is approximately Π cycles, and a long lifetime is greater than Π (sometimes many Π) cycles. Long lifetimes typically arise when a value is produced and consumed in different iterations of the application’s loop, or by constant values in the loop.

Due to the cyclic nature of the modulo schedule, storing a long-lived value in a modulo scheduled architecture is challenging, particularly when using structures such as traditional register files. If the address of a storage location is statically determined, then the distance between a register write and subsequent read is at most Π cycles in a modulo schedule. Otherwise the next write in the schedule will clobber the current value.

Rau et al. [6] and Dehnert et al. [7] developed the Cydra-multiconnect rotating register file, for VLIW processors with modulo schedules. They showed that by rotating the address of the read and write ports it was possible to create a single code sequence for loop bodies and allow values to live longer than Π cycles. This reduced the size of executable code segments and simplified the compiler code generation.

Despite its advantages, few CGRAs with module schedules make use of the rotating register file. One notable exception is the ADRES architecture [8], [9], which describes having rotating register files but does not evaluate their advantages or overhead separately from other architectural features. The RaPiD [10] architecture also provided a rotating register file using a similar mechanism, but with an explicit shift command.

A. Value Lifetime Analysis

Figure 1 shows the histogram of value lifetimes for our benchmarks. We see that 58% of the values have a lifetime of at most 2 cycles, and 79% have a lifetime of at most 4 cycles. Furthermore, the percentage of values with a given lifetime rapidly approaches zero after a lifetime of 4. Overall the average of the first 90% of value lifetimes is 3 cycles, and the remaining 10% have an average lifetime of 24 cycles. Given such a distribution we expect that a heterogeneous collection of

storage structures, some optimized for short-lived values and others for long-lived values, will be better than a single universal storage structure.

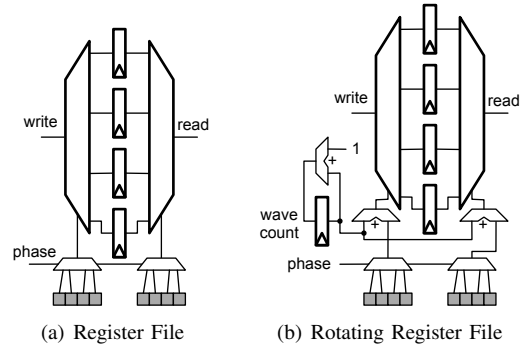


Figure 3. Register file structures composed of 4 registers.

III. STORAGE STRUCTURES

There are several choices for storing values in a CGRA: memories, register files, shift registers (or FIFOs), pipeline registers, and individual registers distributed throughout the system. In Figures 2 and 3 several of these structures are shown with 4 registers and one input and output port. The grey boxes are configuration memory cells, which represent the configuration words for a CGRA’s modulo schedule. These configuration words control multiplexers in all of the designs, as well as register enables in 2(b) and 2(c). These structures represent the majority of the desired design space for collecting and composing registers.

Table I shows the area and energy characteristics for each register structure; the methodology for generating these characteristics is presented in §IV-A. Dynamic energy is reported as energy per write and read plus the recurring energy to store a value for multiple cycles. Write or read energy is accrued for each bit of a value plus a fixed amount per operation. The final column is the longest that a value can be stored in a structure.

A. Retiming Chains

Retiming chains were used in both HSRA [11] and RaPiD [10], for input and output retiming, to buffer

| Structure Type | Abrv. | Area (μm^2) | Static energy per clock | Energy per | | Recurring energy per value | Max. Lifetime |
|--------------------|-------|-----------------------------|----------------------------|--------------------------------|-------------------------------|---|------------------|
| | | | | write | read | | |
| Rotating Reg. File | RRF | 1946.9 | 132.8 | $366.7 + 29.6 * \#\text{bits}$ | $329.7 + 5.7 * \#\text{bits}$ | $\frac{25.5 * \#\text{waves}}{\#\text{values}}$ | $4 * II$ |
| Register File | RF | 1722.8 | 130.3 | $293.3 + 29.6 * \#\text{bits}$ | $256.3 + 5.7 * \#\text{bits}$ | 0 | II |
| Retiming chain | RT | 1580.0 | 82.0 | $10.4 * \#\text{bits}$ | $256.3 + 5.7 * \#\text{bits}$ | $(10.4 * \#\text{bits} * \#\text{cycles})$ | 4 |
| Shift register | SR | 1738.5 | 177.3 | $10.4 * \#\text{bits}$ | $256.3 + 5.7 * \#\text{bits}$ | $(10.4 * \#\text{bits} * \#\text{shifts})$ | $4 * II$ |
| Dist. Reg. | DistR | 292.0 | 30.0 | $10.4 * \#\text{bits}$ | 0 | 0 | II |

Table I

AREA AND ENERGY METRICS FOR EACH TYPE OF REGISTER BLOCK (EACH WITH 4 REGISTER ENTRIES) AND A SINGLE DISTRIBUTED REGISTER. ENERGY IS REPORTED IN FEMTOJOULES (FJ) AND BASED ON THE NUMBER OF BITS THAT CHANGE IN THE STRUCTURE.

values with short lifetimes that travel directly between functional units. A retiming chain, illustrated in Figure 2(a), is a sequence of registers with one input and one output and a multiplexer that is used to extract the value at different stages of the chain. Values advance through the chain on each clock cycle, making the hardware very simple. In our implementation, unused registers in retiming chains are statically clock gated to conserve energy. In Table I we see that the retiming chain is the smallest structure, consumes the least amount of energy per write, and has the lowest static energy. However, it also has the highest recurring energy, which results from transferring the contents of one register to the next on each cycle. Finally, the longest that a value can be stored in it depends solely on its length.

B. Shift Registers

Figure 2(b) shows a shift register – essentially a retiming chain with a common register enable to control the shifting of values. We use a specialized version called a variable shift register, Figure 2(c), that dynamically determines how many of the registers to shift and how many to hold. This is achieved by using individual configuration bits for each register’s enable, rather than a shared enable signal. On each phase of execution the shift register is divided into two contiguous regions, the fore region that shifts data and the aft region that holds its current data. The split point for these regions varies from cycle to cycle and either region can be empty. By varying the size of the aft region, the lifetime that a value can reside in the shift register is substantially improved. Additionally, the variable shift register is more amenable for use with the SPR CAD tool.

Unlike the retiming chain, the shift register does not have to shift a value through the chain on every clock cycle. Furthermore with the variable shift register, values in the aft portion of the structure do not have to shift when new values are pulled in to the system. This feature reduces the energy consumed and allows a value to be stored for longer in the shift register than

the retiming chain. However, as the utilization increases, the values are forced forward in the chain, and its power consumption becomes similar to the retiming chain.

C. Register Files

Register files allow greater flexibility for reordering values than either retiming chains or shift registers. As shown in Figure 3(a) they use multiplexers on both read and write ports to steer values to registers. A difference between both register files and the retiming chain or shift register is that register files have a significant energy cost per write. This cost is a result of the decoder and initial fanout to all registers in the structure versus only writing to the head of the retiming chain or shift register. As a result, to make a register file competitive against either the retiming chain or shift register a value has to live in the register file long enough to amortize the cost of the write. This is the primary reason that register files are a poor choice for storing short-lived values.

D. Rotating Register Files

The rotating register file enhances a normal register file for use with modulo schedules. Figure 3(b) shows the addition of a wave counter and register that is used as an offset for the read and write address. The wave counter is incremented on each pass through a modulo-schedule. The offset acts like automatic register renaming between physical and logical locations in the register file. This simplifies the access pattern to the register file for values that live in the register file longer than the length of the modulo schedule.

For values with a medium life span (approximately II cycles) both the register file and the rotating register file perform similarly, consuming dynamic energy only on read and write. For values with a lifetime that is longer than the II , the rotating register file has a distinct advantage over the traditional register file. To store a long-lived value the register file has to read out the value and re-write it to a different location. This process will

typically incur a trip across the cluster’s crossbar. In contrast, the rotating register file will simply shift its address offset counter so that the next iteration of the value does not overwrite the current instance.

E. Storage Structures Synopsis

A priori we can surmise the following advantages and disadvantages for each structure. The rotating register file and shift register are more sophisticated versions of the register file and retiming chain; they can store values longer but have a small additional area and energy overhead. Register files have a disadvantage because the initial energy required to write into them make them a poor choice for short-lived values and they are unable to hold long-lived values. The retiming chain, which is the simplest possible structure, would win based on area-only analysis, but the cost of moving data on every cycle means that the dynamic energy cost will exceed its static energy and area advantages.

The remainder of this paper describes our experiments aimed at determining the best composition of storage structures for CGRAs. We seek to answer the question of how many of these different structures should there be, and how should they be distributed throughout the architecture.

IV. BACKGROUND

This work is part of the larger Mosaic project as detailed in [12], [13]. The Mosaic project is exploring architecture, mapping algorithms, compilers, and programming language issues related to CGRAs. We have developed a complete toolchain that allows us to take applications written in a high level language (Macah), map them to families of CGRA architectures, simulate them, and collect area, energy, and performance results for each mapping. In the Mosaic toolflow, a CGRA’s embedded memories are exclusively managed by the application, while all of the registers and register structures are allocated by SPR, the place and route tool. The flexibility and extensibility of the SPR CAD tool [12] was instrumental to analyzing the storage structures of interest. Although these experiments were conducted on a CGRA architecture, we believe that they apply more broadly to other styles of spatial architectures.

In previous work [13] we have explored the advantages of scheduled versus statically configured interconnect for CGRAs in a grid topology and the advantage of a dedicated 1-bit control network that augments a word-width datapath. This paper builds upon these results, using a partitioned interconnect with a fully scheduled,

32-bit, word-wide datapath and a fully static single-bit control path. The methodology for modeling and simulating the interconnect is the same as in [13].

Our experiments use the same benchmark suite used in [13]. Each alternative architecture is evaluated by running each benchmark with four different placement seeds. Note that we size each application and target architecture so that no resource in the array is utilized more than 80%. Typically the area-energy-delay product is the best metric for evaluating these architectural tradeoffs, but each architecture template is optimized to provide nearly equivalent application performance. This lets us simplify the analysis to use just the area-energy product for comparison. We hold performance constant for each application by forcing SPR to use the application’s minimum II across architectures and running architectures at similar clock frequencies. The application-architecture mapping with the lowest total energy was selected and used to compute the area-energy product for that benchmark. Table II provides a summary of architectures tested and improvements in area-energy product versus the baseline architecture.

A. Energy, Area, and Performance Evaluation

To provide an evaluation framework for our architecture template we have created a simulation environment within the VCS Verilog simulator that tracks bit transitions for each signal and the activity of physical structures such as rotating register files. The physical models were created using full custom design and memory compilers in a 65nm process. Energy and delay statistics were obtained from SPICE simulations and a commercial memory compiler.

V. BASELINE CGRA ARCHITECTURE

The baseline coarse-grained reconfigurable array architecture that we study is a 2-D array of repeated tiles, shown in Figure 4(a). Each tile consists of a compute cluster and a switchbox that connects to a global grid interconnect. Each compute cluster is composed of 4 processing elements, each of which contains an arithmetic functional unit and possibly some local storage. Figure 4(b) highlights the logical connectivity of a cluster. The baseline architecture has separate 32-bit and 1-bit components that include functional units, crossbars, and interconnect. The functional units retime values on their inputs rather than their outputs, which was shown by Sharma et al. [14] to provide a 19% improvement in area-delay product.

The 32-bit components comprise four arithmetic and logic units, two embedded memories, register storage,

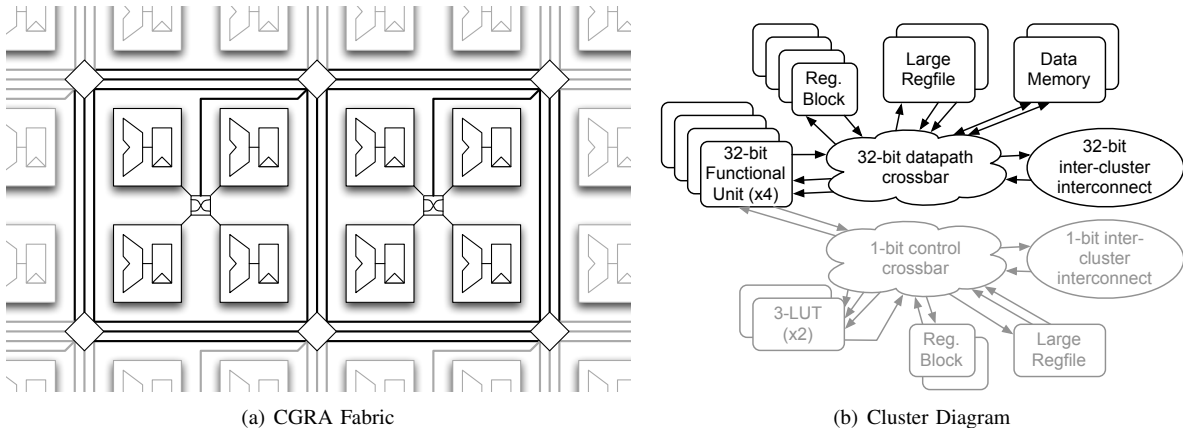


Figure 4. Block diagrams of CGRA with clusters of 4 PEs connected via a grid of switchboxes and an individual cluster with 32-bit datapath in black and 1-bit control path in grey.

and possibly send and receive stream I/O ports. The embedded memories are used for storage that is explicitly managed by the application, and not by the SPR compiler. Therefore they behave as simple value producers and consumers. The 1-bit components comprise two 3-LUTs, and register storage that has the same structure as the 32-bit datapath.

Register storage in split into short-term and long-term storage structures. The short-term storage structures contain a small number of registers and are generically referred to as register blocks. Register blocks have 4 registers each because they are designed to store the short-lived values, the majority of which have a lifetime of 4 or less. The large register files, shown in Figure 4(b), provide long-term storage; they have 16 entries each, enough to hold long-lived values and constant values.

VI. MANAGING LONG-LIVED VALUES

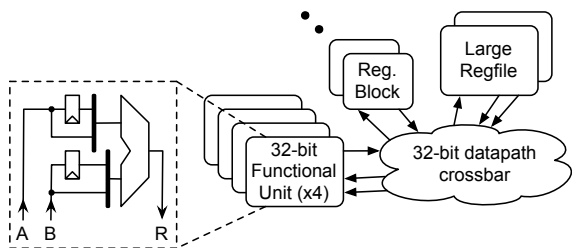


Figure 5. Simplified diagram of 32-bit datapath for baseline architecture and functional unit with pipeline registers for input retiming.

The first experiment examines the effect of using rotating register files for the long-term storage instead of traditional register files. Traditional register files are a common storage structure for spatial accelerators, and serve as the baseline long-term storage for these

experiments. Figure 5 shows a simplified diagram of the baseline architecture, with two large, 16-entry register files that provide the primary long-term storage. They are directly attached to the cluster’s crossbar, and shared between the functional units. The functional units use pipeline registers for input retiming. The experiments were conducted with each type of storage structure for the register block and a summary of the results are shown in the first group of Figure 6-VI.

Replacing traditional register files with rotating registers reduced the area-energy product by 19%, when using two distributed registers for short-term storage. Across all types of register blocks, there was an average reduction in area-energy of 13%. This occurs because a traditional register file is unable to store a value for longer than II cycles, and so long-lived values have to go through the crossbar to reenter the register file. On average a value spent 11.3 cycles in the large rotating register files and only 3.2 cycles when using traditional large register files. The average II for our benchmarks was 3.9 cycles.

VII. STYLE OF SHORT-TERM STORAGE

We described in §III several different types of register blocks that can be used to store short-lived values. Shift registers and retiming chains offer cheap storage of short-lived values, though with less flexibility than register files or distributed registers. This experiment uses the baseline architecture, shown in Figure 5, with two large rotating register files for long-term storage. In this experiment we vary the type and quantity of the register blocks in Figure 5.

Figure 6-VII shows that the best area-energy product is achieved by using two distributed registers; *i.e.* two register blocks each containing a single register. However, using two rotating register file blocks is almost

| Arch. Name | # Dist Regs. | # Reg Blocks | Reg Block Location | Input retiming | Feedback | Large RF | Private Const. | Improvements in area-energy vs. Base VI |
|------------|--------------|-----------------|--------------------|----------------|------------|-----------------|----------------|---|
| Base VI | 2 | 0 | cluster | pipeline | none | 2x RF | no | 1.0× |
| Opt. VII | 2 | 0 | cluster | pipeline | none | 2x Cydra | no | 0.81× |
| Alt. VII | 0 | 2x 4-RRF | cluster | pipeline | none | 2x Cydra | no | 0.82× |
| Opt. VIII | 1 | 0 | cluster | enabled | none | 2x Cydra | no | 0.77× |
| Alt. VIII | 0 | 1x 4-RRF | cluster | enabled | none | 2x Cydra | no | 0.77× |
| Opt. IX | 0 | 0 | cluster | enabled | yes | 2x Cydra | no | 0.75× |
| Opt. X | 0 | 1x 4-RRF | FU | enabled | yes | 2x Cydra | no | 0.73× |
| Opt. XI | 0 | 1x 8-RRF | FU | enabled | yes | 1x Cydra | yes | 0.69× |

Table II

SUMMARY OF EXPERIMENTS WITH BASELINE, OPTIMIZED, AND ALTERNATE ARCHITECTURES, SHOWING INCREMENTAL AND OVERALL IMPROVEMENTS IN AREA-ENERGY PRODUCT. BOLD ENTRIES SHOW ARCHITECTURAL FEATURES THAT CHANGE FROM ROW TO ROW.

as efficient in terms of the area-energy product, while providing more storage capacity and routing flexibility. The results for using the simpler structures such as retimers and shift chains are at least 1-3% worse than for rotating register files. This is explained by the fact that these simpler structures consume more energy than rotating register files when storing values with intermediate lifetimes.

VIII. USING DYNAMICALLY ENABLED REGISTERS FOR FUNCTIONAL UNIT INPUT RETIMING

The input retiming registers on the functional unit in Figure 5 are fairly inflexible—only holding a value for a single cycle. In this experiment, we investigate the effect of enhancing the input retiming registers with dynamic enables provided by the configuration (similar to the variable shift register), as shown in Figure 7. This allows them to store values for a longer time, reorder incoming values, and reduces the demand for short-term storage (register blocks) when compared to the previous section.

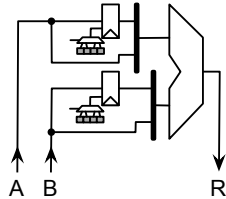


Figure 7. Functional unit with enabled registers for input retiming.

By dynamically enabling the registers in the functional unit, the demand for register blocks reduces. Our testing indicated that the best number of register blocks per cluster reduces from two to one. Figure 6-VIII shows that dynamically enabling the input retiming registers reduces the area-energy product of an architecture with either one rotating register file block or a single distributed register to 0.77× the area-energy of the baseline architecture. Across all register block

styles, using enabled input registers reduced the average area-energy by 4% compared to the architectures in §VII and Figure 5.

IX. ADDING LOCAL FEEDBACK PATHS

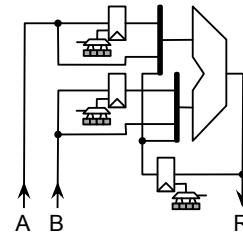


Figure 8. Functional unit with internal feedback.

If a value produced by a functional unit is subsequently used by an operation in the same functional unit (a common occurrence), this value must be routed through the cluster crossbar to a functional unit input register, given the functional units described thus far. In this experiment, the functional units are augmented by an internal, registered feedback path as shown in Figure 8. This path reduces crossbar usage and frees the input registers for storing other values when operations are chained in the same functional unit. On average, this increased static energy by 31%, but it reduced the dynamic energy by 7%.

Adding feedback to the functional unit allowed SPR to successfully route all benchmarks on an architecture with no distributed registers or register blocks. Figure 6-IX shows that this architecture has an area-energy product that is 0.75× the baseline. This is 0.98× the area-energy of the best architecture from §VIII.

X. GROUPING REGISTER BLOCKS WITH FUNCTIONAL UNITS

Instead of associating the registers blocks with the cluster, the register blocks can be grouped with each ALU. This provides more local, private storage than

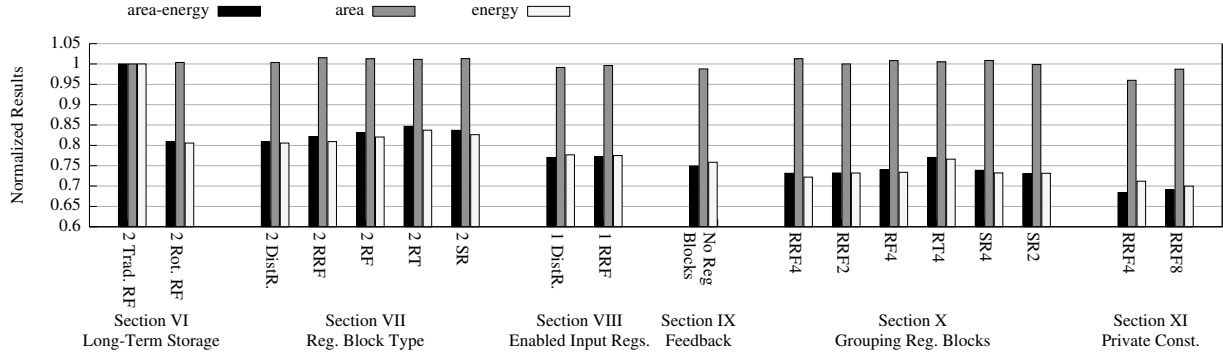


Figure 6. Area, energy, and area-energy results for each experimental section, separated into labeled groups. Functional units for each group are shown in Figures 5, 5, 7, 8, 9, 10, respectively. Each metric is independently normalized to the baseline architecture: §VI and Figure 5.

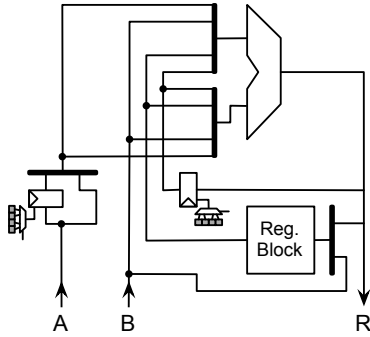


Figure 9. Grouping register block with ALU.

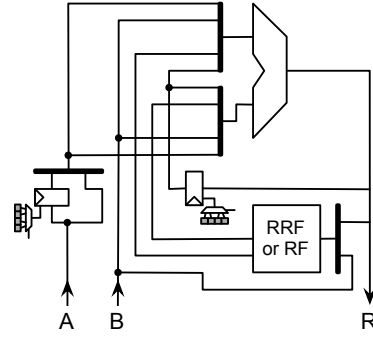


Figure 10. Supporting constant in local register files.

just a feedback path, at the risk of increasing the architecture’s resource fragmentation. For this experiment, we remove the register blocks that were attached to the cluster crossbar and add a register block to each functional unit as shown in Figure 9. This increases the number of register blocks over the best results in the previous section, but decreases the crossbar size and traffic.

Figure 6-X shows the detailed area, energy, and area-energy product for each style of private register block. Using a rotating register file for the register block reduced the area-energy product to $0.73\times$ the area-energy of the baseline architecture and $0.97\times$ the area-energy of the optimized architecture from §IX and Figure 8. To minimize the increase in registers used for the register blocks we tested designs that had only two registers per local register block for architectures with rotating register files and shift registers, shown as RRF2 and SR2 the graphs. The results show very little difference, indicating that reductions in area are offset by increases in energy, as values are forced to use more expensive resources.

XI. SUPPORTING PRIVATE CONSTANT VALUES

In the architectures tested thus far, constants are stored in the large register files since constants are in some ways the ultimate in long-lived values. In this experiment, we study the effect of storing constants in the ALU’s private register blocks when implemented using rotating register files. Constants are stored by partitioning the register file into RAM and ROM regions at configuration time. Partitioning is achieved by selectively disabling the addition of the wave counter to the upper entries of the register file. We increase the register block ports to 2 read and 1 write, shown in Figure 10, but we found that we can also reduce the number of large register files in the cluster from two to one. Since storing many constant values in the private register file may force other values that could be local into the more expensive large register file, we tested architectures with 4 and 8 registers per register block.

The last group of Figure 6-XI shows that supporting constants in a rotating register file results in an area-energy product that is $0.93\times$ the area-energy of an architecture without private constants. The area-energy for a 4-entry register block is $0.68\times$ the area-energy of the baseline architecture. However, the 8-entry register

file is preferable because it doubles the local storage with only a 1% loss in area-energy, for a total area-energy that is $0.69\times$ the area-energy of the baseline architecture. The architecture with 8-entry register blocks consumed less energy than the one with 4-entry register blocks, showing that the increased capacity was able to keep values local, and highlights the benefit of using a medium sized rotating register file for storing values with medium length lifespans.

XII. CONCLUSIONS

In this paper we have considered numerous different mechanisms for storing values in a CGRA. Our experiments show that:

- Rotating register files consistently outperform other structures for storing medium- to long-lived values. Although rotating register files are more expensive in terms of area than other options, the improved energy more than makes up for their expense.
- The energy efficiency of enabled registers is well worth their configuration memory costs for short-lived values.
- While rotating register files and distributed registers often provide the same area-energy product, distributed registers are more area-efficient, while rotating register files are more energy-efficient.
- Normal register files are largely ineffective in modulo-scheduled architectures, since they cannot hold values longer than Π cycles, yet have much greater overheads than other structures for short-lived values.
- It is better to keep values close to the functional units using local feedback and private register blocks, than to use register blocks that can be shared among functional units, but require values to transit the cluster crossbar.

Overall, these observations provide an architecture that is 2% better in area, 30% better in energy, and 31% better in area-energy product than a reasonable baseline architecture.

XIII. ACKNOWLEDGEMENTS

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

REFERENCES

- [1] 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 *Intl. Conf. on FPL*, vol. 2778, Lisbon, Portugal, 2003, pp. 61–70, 2003.
- [2] 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.
- [3] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *IEEE Symp. on FCCM*, 1996, pp. 157–166.
- [4] M. Lam, "Software pipelining: an effective scheduling technique for VLIW machines," in *ACM SIGPLAN Conf. on PLDI*. ACM Press, 1988, pp. 318–328.
- [5] B. R. Rau, "Iterative Modulo Scheduling: An Algorithm For Software Pipelining Loops," in *Intl. Symp. on Microarchitecture*. ACM Press, 1994, pp. 63–74.
- [6] B. R. Rau and C. D. Glaeser, "Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing," in *Intl. Symp. on Microarchitecture: 14th annual workshop on Microprogramming*. IEEE Press, 1981, pp. 183–198.
- [7] J. C. Dehnert, P. Y.-T. Hsu, and J. P. Bratt, "Overlapped loop support in the Cydra 5," in *Intl. Conf. on ASPLOS*. New York, NY, USA: ACM, 1989, pp. 26–38.
- [8] F. J. Bouwens, "Power and Performance Optimization for ADRES," MSc Thesis, Delft University of Technology, August 2006.
- [9] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architectural Exploration of the ADRES Coarse-Grained Reconfigurable Array," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, S. B. . Heidelberg, Ed., vol. Volume 4419/2007, March 2007, pp. 1–13.
- [10] C. Ebeling, D. C. Cronquist, and P. Franklin, "RaPiD - Reconfigurable Pipelined Datapath," in *Intl. Workshop on FPL*, R. W. Hartenstein and M. Glesner, Eds. Springer-Verlag, Berlin, 1996, pp. 126–135.
- [11] W. Tsu *et al.*, "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array," in *ACM/SIGDA Intl. Symp. on FPGAs*. ACM Press, 1999, pp. 125–134.
- [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 Intl. Symp. on FPGAs*. ACM, 2009, pp. 191–200.
- [13] B. Van Essen *et al.*, "Static versus scheduled interconnect in Coarse-Grained Reconfigurable Arrays," in *Intl. Conf. on FPL*, 31 2009-Sept. 2 2009, pp. 268–275.
- [14] A. Sharma, K. Compton, C. Ebeling, and S. Hauck, "Exploration of Pipelined FPGA Interconnect Structures," in *ACM/SIGDA Intl. Symp. on FPGAs*, 2004, pp. 13–22.