Offset Pipelined Scheduling: Conditional Branching for CGRAs

Aaron Wood
Department of Electrical Engineering
University of Washington
Seattle, WA USA
arw82@uw.edu

Scott Hauck
Department of Electrical Engineering
University of Washington
Seattle, WA USA
hauck@uw.edu

Abstract— **Coarse Grained Reconfigurable Arrays (CGRAs) offer improved energy efficiency and performance over conventional architectures. However, modulo counter based control of these devices limits efficiency for applications with multiple execution modes. This work presents a new type of architecture that adds support for branching control flow to CGRAs. The pipelined program counter CGRA framework blends the high parallelism of traditional CGRAs with the flexibility of commodity processors. Offset Pipelined Scheduling (OPS) is the basis of an enhanced CGRA tool chain targeting these devices. OPS is shown to provide an average 1.94x speed up for benchmarks that are resource limited when modulo scheduled.**

Keywords- CGRA, scheduling, software pipelining

I. Introduction

Coarse grained reconfigurable arrays (CGRAs) [1] offer improved energy efficiency and performance compared to commodity architectures. When well utilized, they combine high density, performance and energy efficiency. They embrace features of standard FPGAs while moving away from bit oriented resources towards larger logic blocks and correspondingly wider interconnect. Such architectures are register rich using pipelined interconnect and time multiplexed resources to maximize potential utilization. CGRAs are capable of greater parallelism than commodity processors and have lower area and performance overheads compared to FPGAs.

The time multiplexed control of a CGRA is managed by a global modulo counter. Each counter value selects a configuration of the logic and routing resources on the device to execute in a cyclic sequence. A modulo schedule is well matched to a single loop body, but managing more complex control flow becomes costly. An application consisting of a sequence of loop bodies will have wasted issue slots for predicated operations in addition to phi operations to merge control flow on a CGRA.

Very long instruction word (VLIW) processors avoid this issue by stitching together modulo scheduled blocks into a larger application with additional prologue and epilogue code. This is challenging in a CGRA due to limited instruction memory and difficulty propagating control information across the device. Compared to a VLIW processor, CGRAs have much larger per cycle instruction memory requirements to configure both logic resources and interconnect.

To transition from one mode of execution to another suggests a branching capability comparable to that of a conventional processor. However, filling and draining the execution pipeline can be costly for applications with high latency or relatively few iterations between mode transitions. Moreover, the ability to branch brings the additional difficulty of broadcasting the change in control flow across the device.

In this paper we introduce Offset Pipelined Scheduling (OPS). Our approach provides a branching mechanism to broaden the scope of applications that can be efficiently scheduled on CGRAs. Execution of resources across the device is staggered to provide the time needed to send changes in control flow. In OPS, one resource domain acts as a leader, performing operations from a given mode as well as computing loop or branch conditions to determine the next mode to initiate. Other domains follow the leader, receiving the same program counter value from the leader, but delayed by a fixed offset number of cycles. In this way, the resource domains in the system form a pipeline.

OPS draws inspiration from iterative modulo scheduling (IMS) [2]. It generates schedules that execute in a cyclic fashion similar to IMS. The primary distinction between a modulo schedule and our approach is that a modulo schedule allows operations to be scheduled into time slots modulo the initiation interval, while time slots in OPS represent their specific cycle. Fig. 1 illustrates this key difference. An operation in a modulo schedule might be scheduled at a latency greater than the initiation interval (II). OPS requires that operations be scheduled on an explicit time slot and does so by adjusting time windows of the various resources to achieve this. While this constraint puts OPS at a

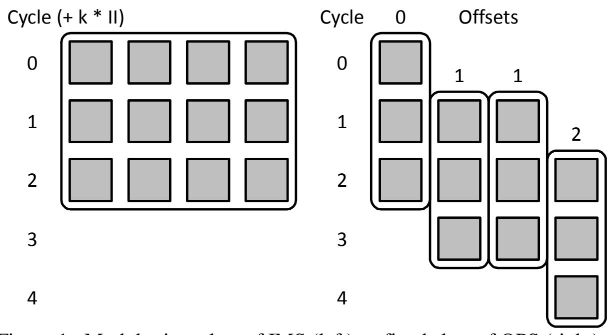

Figure 1. Modulo time slots of IMS (left) vs fixed slots of OPS (right).

disadvantage for a single loop body relative to IMS, it aids in the support of applications with multiple modes. By explicitly staggering compute resources, OPS facilitates the transition between phases of a computation and broadcast of control information.

II. Execution On Branching CGRAs

The following subsections detail features of offset pipelined schedules and the target enhanced CGRA architectures.

A. *Expanding the Scope of CGRA Code*

Software pipelining by modulo scheduling was originally envisioned for VLIW machines where functional units must be statically scheduled to provide maximum utilization. Modulo scheduling for CGRAs is a logical extension with modulo counter control and a focus on inner loop parallelism. In this paper we broaden the scope of application code mapped to CGRAs, including allowing multiple loops to coexist on the same device.

B. *Modes*

A mode is a partition of the target code that will be run exclusively until a transition to another mode. It is a subset of the code mapped to the CGRA. Consider two successive loop bodies which can be logically divided into two separate modes and executed in isolation. Such a distinction matters little in the context of a conventional processor since branching to move between blocks of code is trivial. For a CGRA where each cycle of a schedule is essentially a very long instruction word, efficient utilization of the hardware and program length become important considerations.

K-means clustering provides an example of modal behavior in an application. The algorithm logically breaks into two modes; assigning observations to clusters, and updating cluster centroids. A conventional CGRA must execute both modes on every iteration or face reconfiguring the device at each transition, either way generating a sizeable overhead.

C. *Pipelined Program Counter Architecture*

The key architectural feature leveraged by OPS is the pipelined distribution of control information. Program counters are spread throughout the device, with each receiving its instruction pointers from a neighbor in the array. Each program counter controls a small group of resources, including ALUs, LUTs, memories and interconnect comprising a domain. An application mapped by OPS includes offset assignments for the domains. A domain can be thought of as a small VLIW machine with its own program counter managing controlling resources. OPS harnesses the collection of domains into a single unit to efficiently execute the target code.

D. *Offsets*

The lead domain is defined to have an offset of 0 while other domains have offsets measured relative to the lead. The offset is simply the cycle latency of receiving a program counter value from the lead domain. The leader issues all

```
initializeIIs()
do {
   initializeOffsets()
   resetSchedulingCache()
   do {
      buildOffsetReservationTable()
      if ASAPschedule()
         return SUCCESS
      offsetsUpdated = offsetAdjustment()
   } while (offsetsUpdated)
   iisUpdated = incrementIIs()
} while (iisUpdated)
```

Figure 2. Top level OPS algorithm

control flow decisions which cascade through the domains. The organization of the domain offsets factors into the placement of operations on the device and is beyond the scope of this paper. Each domain is assigned an offset that is shared across all modes of the target application.

III. Offset Pipelined Scheduling

The goal of OPS is to generate a legal, high performance schedule for the target application. There are three components of the resulting schedule:

- Initiation interval assignments for each mode
- Offset assignment for each domain
- Time slot assignment for each operation

Smaller initiation intervals are desirable as this corresponds to a faster application. Offsets are assigned to provide issue slots necessary to cover all operations. Operations must be assigned to available time slots based on the offsets and II assignments. The pseudo code in Fig. 2 outlines the algorithm.

The general approach is to schedule the application using a given II and offset assignment, and then adjust those parameters as necessary to achieve a legal schedule. The IIs and offsets determine the arrangement of issue slots available stored in a structure called the offset reservation table (ORT) an example of which is shown in Fig. 3. Here we have two domains and three modes. The modes have IIs 2, 1 and 3 from left to right and the domain offsets are 0 and 2. Note that for a given domain, the same offset is shared among all modes.

IIs begin at the minimum possible given recurrence or resource constraints and offsets begin at zero. Operations are

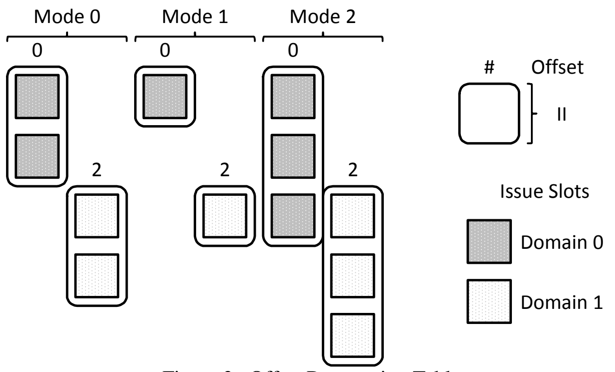

Figure 3. Offset Reservation Table

scheduled into the ORT in an as-soon-as-possible fashion. In general, this process will fail due to some operations lacking legal issue slots. In this case, offsets are increased to make issue slots available where they are needed. If offset adjustment does not lead to a successful scheduling, a mode II is incremented to provide additional slack. The scheduling is successful once all operations have a legal issue slot.

A. *Offset Adjustment*

Offsets are adjusted both through a deterministic shaping process and a heuristic exploration process. In either case, offsets are only allowed to increase for given II settings. Deterministic adjustment moves offsets as late as strictly necessary based on the current schedule. Since all operations are scheduled as-soon-as-possible and offsets begin at zero, when an operation has no issue slot, there must be one vacant earlier in the ORT. Moving this issue slot is accomplished by increasing an offset. More specifically, each offset is examined from largest to smallest. For each mode M, the II_M latest operations are assigned to the domain for all of its resources. By applying this concept to all offsets, a profile of the latest necessary offsets is recovered. This process alone may force operations that were scheduled early to move later to fill a legal issue slot. However, if no deterministic moves can be made, the heuristic approach tries incrementing an offset to ensure progress towards a legal schedule.

B. *Incrementing IIs*

If the offset adjustment process does not yield a legal schedule, OPS falls back on increasing IIs to add flexibility to the scheduling. While IMS only has one II to increment, OPS must choose which mode II will be degraded. Priority information is based on profiling the application. The overhead is the product of mode priority and the ratio of current mode II to the minimum mode II calculated at initialization.

The algorithm prefers to increment the lowest overhead mode since this will minimize the impact on overall application performance. This preference is capped at 2x the overhead of any other mode, heuristically selected to avoid skewing the IIs too far.

IV. Evaluation

Offset Pipelined Scheduling is evaluated in comparison to iterative modulo scheduling. The first section describes the target architecture. The benchmarks are introduced and results are presented to provide insight into OPS behavior.

A. *Architecture*

The target architecture consists of control domains comprised of two 32-bit ALUs, two 4-input LUTs, a memory block, and a stream port. Each domain also contains a program counter unit for managing the flow of execution. The selected CGRA configuration is based on optimized architectures developed in [3]. This paper focuses strictly on scheduling; placement and routing will be considered in future papers.

TABLE I. Applications

Application	Description
Bayer	Bayer filtering, includes threshold and black level adjustment
DCT	8x8 discrete cosine transform
DWT	Jpeg2000 discrete wavelet transform
K-means	K-means clustering with three channels and eight clusters
PET	Positron emission tomography event detection and normalization

While the target architecture is designed for use with OPS, it is also used for the modulo scheduling baseline. In this case, the device has the same resources but is configured to mimic a simple modulo counter.

B. *Benchmarks*

The applications used for evaluation are listed in Table 1 with a brief description. They represent a cross section of signal processing algorithms typically targeted to CGRAs. In order to compare performance between the OPS and IMS implementations, the cycles are normalized to the recurrence limited cycles of the corresponding IMS implementation. This provides insight into the performance of OPS relative to IMS and allows the applications to be compared to each other.

C. *Results*

Fig. 4 summarizes the results showing the ratio of IMS to OPS normalized cycles to provide a speed-up metric. The geometric mean is also included, aggregating across all applications. As we provide more resources, the IMS implementation eventually reaches its recurrence limit and OPS provides no additional benefit in terms of performance. OPS can achieve the same performance with fewer resources, or better performance can be achieved with the same number of resources when operating in a resource limited area of the curve.

The applications eventually reach their recurrence limit with enough resources. In these cases, despite unused issue slots, the application can still be scheduled for maximum performance. However, for the various device sizes where

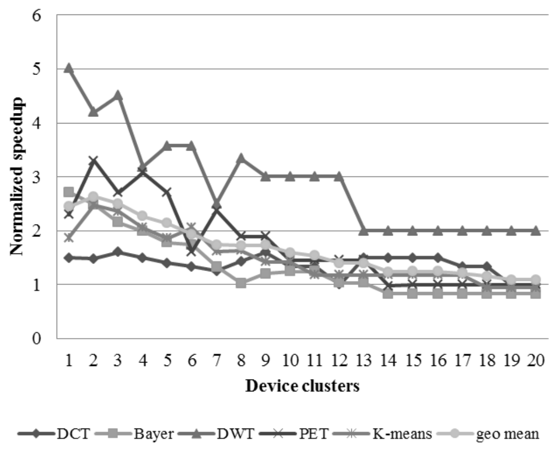

Figure 4. IMS vs OPS performance summary

applications are resource limited, OPS provides an average speed up of 1.94 times over a modulo scheduled solution.

OPS provides significantly better performance when resources are limited, but when only considering scheduling there comes a point where IMS can reach the same performance as OPS (the intrinsic recurrence loops). Since this IMS solution would be significantly larger than the corresponding OPS solution, IMS will likely have greater difficulty when placement and routing are considered. Those steps will be addressed in future work.

It is also important to note that the hardware needed to support IMS and OPS schedules are relatively similar. In fact, an OPS device can be converted to run full-fledged IMS schedules simply by forcing all domains to perform a single loop. Thus, a single design suite could use OPS for resource-constrained modal computations, and use IMS for single-mode designs, or designs without resource constraints.

OPS runtime did not exceed approximately 10 seconds for any of the schedules generated in this work.

V. Conclusion

When considering word-oriented FPGAs and FPGA-like systems, architectures have split into massively parallel processor array (MPPA) and CGRA style devices. CGRAs provide a huge amount of parallelism, and have automatic mapping tools that can spread a computation across a large fabric, but their restriction to modulo-counter style control significantly limits their ability to support applications with more complex control flow. MPPAs provide an array of full-fledged processors, with a great deal of fine-grained parallelism, but they are much less tightly coupled than CGRAs and generally must be programmed via explicitly parallel programming techniques.

In this paper we have taken a step toward merging these two styles of devices. By providing a new program counter model that keeps communication local yet can support more complex looping styles, we can support a much richer set of applications. The integration of conditional branch and complex control flow operations significantly increases the computational density and range of target applications, supportable by these systems.

The scheduling algorithm for these devices automatically schedules issue slots, determines individual domain offsets, and sets mode IIs to achieve a high-performance and dense implementation. In future work we will extend these efforts to include placement and routing tools for this new style of coarse-grained computational resource.

VI. Related Work

Related architectures primarily fall into two categories; MPPAs and CGRAs. MPPAs such as Ambric [4] and Tilera [5] are composed of discrete processors. They are less tightly integrated than the proposed CGRA and must be programmed using traditional parallel programming techniques.

CGRAs such as Mosaic [6] and ADRES [7] are designed for modulo scheduled execution. These architectures are tightly integrated and offer tool support [8] to leverage the array but are limited to modulo counters for control.

The Tabula [9] SpaceTime architecture is a commercial product similar to a CGRA using a modulo counter mechanism for time multiplexing. However, these devices are fine grained and provide a conventional FPGA tool chain abstraction for the underlying hardware.

A. *Mapping Strategies*

Algorithms for mapping applications to CGRAs have been based on compiler tools for VLIW and FPGA architectures. Scheduling specifically draws from modulo scheduling work treating the device as a large VLIW style processor. Modulo scheduling and IMS [2] in particular inspire the software pipelined schedule and iterative nature of OPS. Modulo scheduling with multiple initiation intervals [10] explores more flexible execution similar to OPS. This earlier work targets a more traditional VLIW machine with a single program counter, very different from the goal of OPS to help automate mapping to multiple control domains on an enhanced CGRA.

Acknowledgment

NSF grant #1116248 supported this work. Special thanks to Suzanne Wood for development of the K-means benchmark.

References

[1] 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

[2] B. R. Rau. Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops. In International Symposium on Microarchitecture, pages 63–74, 1994.

[3] Brian Van Essen, Improving the Energy Efficiency of Coarse-Grained Reconfigurable Arrays, Ph.D. Thesis, University of Washington, Dept. of CSE, 2010. http://www.ee.washington.edu/people/faculty/hauck/publications/dissertation-vanessen.pdf

[4] M. Butts, A. Jones, and P. Wasson. A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing. In Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'08), pages 55-64, April 2008.

[5] Tilera. http://www.tilera.com/

[6] Allan Carroll, Stephen Friedman, Brian Van Essen, Aaron Wood, Benjamin Ylvisaker, Carl Ebeling,Scott Hauck, "Designing a Coarse-grained Reconfigurable Architecture for Power Efficiency", Department of Energy NA-22 University Information Technical Interchange Review Meeting, 2007.

[7] 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 , volume 2778, pages 61–70, 2003.

[8] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, S. Hauck, "SPR: An Architecture-Adaptive CGRA Mapping Tool", ACM/SIGDA Symposium on Field-Programmable Gate Arrays, pp. 191-200, 2009.

[9] Tabula. http://www.tabula.com/

[10] Warter-Perez, N.J.; Partamian, N., "Modulo scheduling with multiple initiation intervals," Microarchitecture, 1995., Proceedings of the 28th Annual International Symposium on , vol., no., pp.111,118, 29 Nov-1 Dec 1995.