

# Design Space Exploration of Instruction Schedulers for Out-of-Order Soft Processors

Kaveh Aasaraai, Andreas Moshovos

*Electrical and Computer Engineering Department, University of Toronto*

*Toronto, Ontario, Canada*

{aasaraai, moshovos}@eecg.toronto.edu

**Abstract**—This work explores instruction scheduler designs for single-issue, out-of-order soft processors targeting irregular workloads. It shows the effect of scheduler size, scheduling policy and back-to-back scheduling on performance, area, and frequency. It is shown that for a modern, high-end FPGA (Altera Stratix III) the best performance is achieved by a small, 4-entry instruction scheduler with an age-based instruction selection policy and back-to-back scheduling. A combined scheduler and register renamer is shown to operate at 303MHz.

## I. INTRODUCTION

Embedded systems are increasingly using FPGAs for the low production cost, flexibility and adequate performance they provide. Such systems often incorporate soft processors as the resulting designs are easier to develop, debug, and modify compared to custom-logic implementations. Depending on the target workload and cost constraints, soft-processors may use one of several architectures with Pipelining, Superscalar [1], Very Long Instruction Word (VLIW) [1], Single Instruction Multiple Data (SIMD), and Vector execution [1], [2] being the commonly used architectures. VLIW, SIMD, and Vector execution exploit programmer- or compiler-extracted instruction-level parallelism (ILP) and shine in the absence of irregular control flow and data accesses. While many workloads exhibit regular data and control patterns, as applications evolve it is natural to expect that other less regular workloads will need to be implemented on FPGAs. For such workloads, superscalar processors that can exploit unstructured and difficult to extract parallelism in programs are more appropriate [1]. A superscalar processor can execute independent instructions in parallel as long as they are adjacent in program order. Out-of-order (OoO) architectures can extract ILP even among non-adjacent instructions.

Recently, it has been shown that single-issue, out-of-order execution has the potential of extracting the same level of parallelism as a four-way superscalar processor [3]. However, OoO architectures require several specialized units that have been developed for custom implementation. It remains an open question whether current OoO architectures can be implemented reasonably over an FPGA fabric. As a first step toward such an implementation, a high-performance, FPGA-friendly register renaming unit has been presented [3].

As an additional step toward an FPGA-friendly, single-issue OoO design, this work studies instruction scheduler implementations. The instruction scheduler, the core of OoO

implementations, is where instructions wait for all their source operands and execution resources to become available. This work starts with a conventional, content-addressable-memory-based scheduler design [4] and studies its implementation on a modern FPGA. Specifically, performance and area are studied as a function of the number of scheduler entries, the inclusion of back-to-back scheduling support, and the use of age-based priority scheduling. Considering the scheduler in isolation it is shown that the best performance is achieved with a two-entry scheduler without back-to-back scheduling and with the simpler location-based selection policy. However, considering part of the rest of the pipeline it is shown that best performance is achieved with a four-entry scheduler with back-to-back scheduling and age-based selection. This four-entry configuration is inexpensive and fast. It uses 164 ALUTs and operates at 303MHz.

## II. INSTRUCTION SCHEDULING

An OoO processor can execute instructions in any order that does not violate data dependencies. Instructions enter the *instruction scheduler*, a pool where they wait until they become *ready*, that is until all their source operands are available. The instruction scheduler identifies ready to execute instructions, then among those, selects  $W$  instructions to issue to functional units,  $W$  being the processor's width. This work focuses on single-issue schedulers ( $W = 1$ ) as past work has shown that it is the number of datapaths that dominates area and frequency on FPGA implementations [3].

An instruction scheduler comprises a wakeup unit and a selection unit. *Wakeup* is responsible for identifying ready to execute instructions among those residing in the scheduler pool. It observes instructions as they produce their results, notifying waiting instructions accordingly. A waiting instruction becomes ready when all its source operands have been produced. All ready instructions request execution from the *selection* unit, which grants execution to those selected for execution.

## III. CAM-BASED SCHEDULER

CAM, the common scheduler design, is based on content addressable memories [4]. Figure 1 depicts CAM's structure. The wakeup part is an array with one row per instruction. Each row contains one column per source operand. Each column contains the source operand tag along with a ready

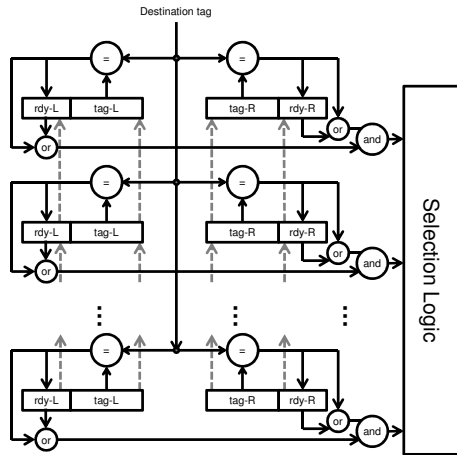


Fig. 1. CAM Scheduler with back-to-back scheduling and compaction. OR gates provide back-to-back scheduling. The dashed gray lines show the shifting interconnect which preserves the relative instruction order inside the scheduler for age-based policy. The selection logic prioritizes instruction selection based on location, i.e., it is a priority encoder.

bit indicating the operand’s availability. For the Nios II ISA targeted in this work, every instruction can have up to two source operands. Each row is accompanied by two comparators. When an earlier instruction finishes execution, its destination register tag is broadcast over all entries and compared to source operands. All matching entries mark their corresponding source operands as available. All instructions that have both their source operands marked ready request execution. The selection logic selects one among those ready instructions for execution. Figure 1 shows the ready signals as inputs to the selection logic.

#### A. CAM on FPGAs

Despite CAM’s simple structure, it is expensive to build on FPGAs. As Section IV shows, area and frequency degrade as the number of entries increases. By increasing the number of entries, the network connecting comparators and source operand tags becomes more complex leading to longer critical paths, hence lower frequencies. Area-wise, as all entries are used for comparison in every clock cycle, block rams cannot be used for storing the tags due to read/write port limitations. Additionally, there is a comparator for each source operand of every instruction resulting in a high resource usage.

#### B. CAM Performance

It is well documented that the ILP that can be extracted increases with the number of scheduler entries, e.g., [4]. The resulting IPC benefits tend to level off after a certain number of entries. The actual saturation point varies depending on the processor architecture and system properties, e.g., memory latency. Actual performance depends not only on IPC but also on the operating frequency. It has been shown that scheduler frequency deteriorates with the number of entries [4]. As a result there is a trade-off between scheduler size and performance in conventional CAM implementations. This work studies this trade-off for FPGA-based implementations.

#### C. Back-to-Back Scheduling

To exploit more ILP it is desirable to execute dependent instructions in consecutive cycles, or *back-to-back*, avoiding bubbles in the pipeline. In this regard, CAM needs to generate ready signals in the same clock cycle as the destination tag is broadcast by earlier instructions. Figure 1 depicts a CAM with back-to-back scheduling. The OR gates ensure that ready signals are produced by either the ready register bit or by the result of the comparisons performed in the current clock cycle. In this design the wakeup and select units must operate in the same clock cycle. Although supporting back-to-back scheduling increases processor IPC, it adversely affects operating frequency. Section IV shows that back-to-back scheduling increases latency. The low clock frequency can overshadow any IPC advantage back-to-back scheduling has to offer. Adding the OR gates has a negligible area overhead as shown in Section IV.

#### D. Scheduling Policy

In the event that more instructions are ready than the available execution units, the Selection unit, based on a selection policy, determines which instructions to execute. This policy can be based on various parameters such as instruction age or location inside the scheduler. Previous work has shown that a selection policy based on instruction age tends to perform better than other, simple to implement heuristics.

One way to consider instruction age in the selection policy is to organize the scheduler as a FIFO queue. FIFOs preserve instruction ordering and provide relative age information based on the location in the queue. Using FIFOs inserting instructions is a trivial queue push operation. In order to remove instructions from arbitrary positions inside the scheduler once they execute, compaction has been implemented in commercial designs to maintain FIFO ordering [5]. In Figure 1, the interconnect among rows provides compaction capability. Upon scheduling an instruction, all entries starting from its position to the bottom (younger) are shifted towards the top (older). This ensures that at any point in time older instructions are placed at the top. This design guarantees that an instruction’s relative position also reflects its relative age. The selection logic uses a priority encoder which prioritizes based on instruction location, giving entries closer to the top higher priority. Compaction impacts area and latency.

The simplest to implement alternative to the age-based policy is location-based scheduling where priority is given to instructions according to where they are stored in the scheduler. In this design the selection logic becomes a priority enforcer with statically assigned priorities, i.e., entries closer to the top have higher priority. Upon scheduling an instruction, its position is marked as free and can be filled with future instructions. Over time, instruction location provides almost no information about its relative age.

## IV. EVALUATION

This section compares the aforementioned scheduler designs based on their area, operating frequency, IPC, and overall

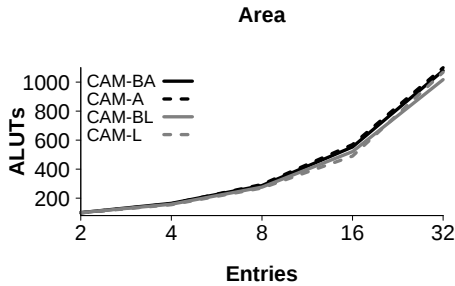


Fig. 2. Number of ALUTs used by scheduler designs.

performance, measured in instructions per second (IPS).

#### A. Methodology

All schedulers were implemented in Verilog and synthesized with Quartus II v9.0 SP1 on the Altera Stratix III FPGA. We have developed a cycle-accurate Nios II full system simulator, booting and running the uCLinux operating system [6] to estimate execution performance. The simulated OoO processor consists of seven pipeline stages, uses 32KB direct mapped caches, and a 512-entry bimodal branch predictor. We simulate 2- to 32-entry instruction schedulers. The DDR2 memory latency is estimated at 20 cycles.

We use benchmarks from the SPEC CPU 2006 suite. This suite is typically used to evaluate the performance of desktop systems [7]. We use these benchmarks as representative of applications that have unstructured ILP assuming they are a reasonable approximation of applications may be ran on future embedded or FPGA-based systems. Measurements are taken for a sample of one billion instructions, after skipping initialization (several billion instructions).

We use the following notation: CAM-B and CAM refer to schedulers with and without back-to-back scheduling, regardless of their entry count. CAM-[B]A and CAM-[B]L refer to schedulers with age- and location-based selection respectively.

#### B. Area

Figure 2 shows how the area of the various designs scales as a function of entry count. Area requirements grow at least linearly with the number of entries. Back-to-back scheduling and selection policy have negligible impact on area. For example, a 4-entry CAM-BA uses 164 ALUTs while CAM-A uses 161 ALUTs. The area scaling is very different than conventional, custom logic implementations that are wire-dominated. Our conclusion is that beyond eight entries the area cost is too high compared to the overall area budget a fast soft processor typically has, i.e., about 1500 ALUTs [9]. Additionally, we conclude that area is primarily determined by the number of scheduler entries.

#### C. Frequency

Figure 3 reports the maximum frequency achieved by each design. Schedulers without back-to-back scheduling consistently reach higher frequencies. For example, the 8-entry CAM-A and CAM-L operate at 344MHz and 404MHz respectively, while the same size CAM-BA and CAM-BL operate

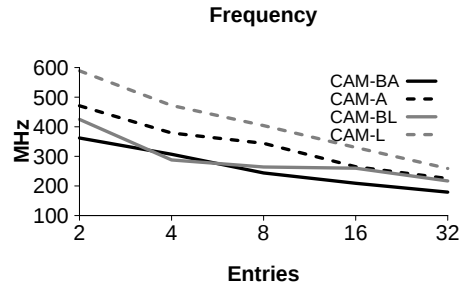


Fig. 3. Maximum clock frequency of the scheduler designs.

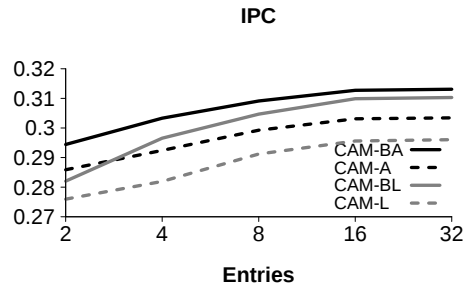


Fig. 4. Instructions per cycle achieved using four scheduler designs.

at 244MHz and 264MHz, a difference of 29% and 35% respectively. We also observe frequency losses by moving from location- to the age-based policy. This drop is highest at 20% between the 16-entry CAM-A and CAM-L which operate at 330MHz and 265MHz respectively.

#### D. IPC

A lower frequency design is not necessarily a worse performing design. Performance depends also on the number of instructions retired per cycle (IPC). Figure 4 reports IPC for the various schedulers. CAM-BA consistently outperforms the rest of the schedulers. The highest difference observed is 7.5% between the two-entry CAM-BA and CAM-BL schedulers. Back-to-back scheduling improves IPC as expected. CAM-BA and CAM-BL are superior to CAM-A and CAM-L respectively. Similarly, the age-based selection outperforms the location-based selection. Most of the IPC benefits come from back-to-back scheduling rather than from age-based selection. CAM-BL consistently outperforms CAM-A beyond the scheduler entry count of two. We conclude that to improve IPC we need to have a scheduler with age-based selection and back-to-back scheduling. However, should any of these features need to be sacrificed (e.g., due to frequency constraints), we find it best to substitute age-based policy with location-based policy rather than removing back-to-back scheduling support. The IPC advantage that back-to-back scheduling provides is greater than that of the age-based selection policy.

#### E. Performance

This section compares the overall performance of various scheduler designs in terms of instructions per second (IPS). IPS considers both clock frequency and IPC. Figure 5 compares the IPS of 2- to 32-entry schedulers. We observe that

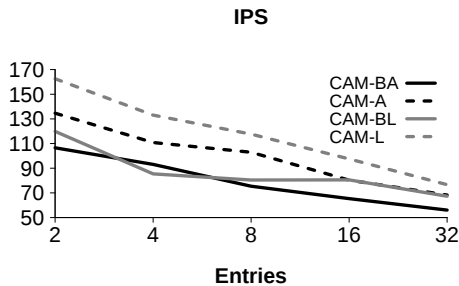


Fig. 5. Overall performance as million instructions per second of four scheduler designs.

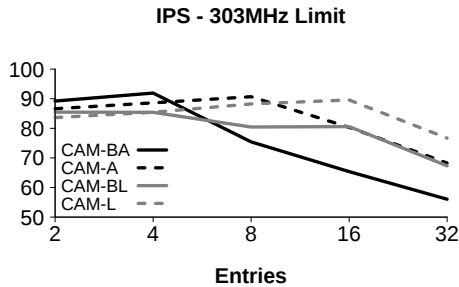


Fig. 6. Overall performance of scheduler designs when the operating frequency is limited to 303MHz.

schedulers without back-to-back scheduling are consistently better performing for all entry counts. Although these designs were shown to reach lower IPCs, their superior clock frequency provides higher overall performance. Similarly, dropping age-based selection in favor of the simpler location-based selection results in higher performance. Increasing the number of scheduler entries reduces performance. Assuming that the entire processor can operate at the scheduler speed, one would conclude that a very small scheduler with two entries would be best.

From previous work we find that an FPGA-friendly renaming unit, a crucial OoO component, operates at 303MHz when implemented on our platform [3]. Thus, in Figure 6 we study the effect of limiting the processor clock frequency to 303MHz. In this case using a slightly larger scheduler proves to be better. The four-entry CAM-BA, eight-entry CAM-A and 16-entry CAM-L are the top three designs. Comparing these three designs we observe that the performance loss due to decreasing scheduler entry count is effectively compensated by the age-based selection policy and back-to-back scheduling support. Additionally, as lower entry counts are desirable considering area usage, we conclude that the 4-entry CAM-BA is the best configuration to choose both in terms of area and performance.

## V. RELATED WORK

To avoid low frequency and high area usage of content addressable memories, Mesa-Martinez et. al. [10] propose SEED, Scalable Efficient Enforcement of Dependences. SEED uses indexed tables to track instruction dependencies. It uses multi-banked structures and is shown to scale well on ASICs.

However, SEED's scalability is shown to be poor on FPGAs as routing overhead among multiple components becomes critical. Fytraki and Pnevmatikatos [11] and Derek et. al. [12] implemented parts of an OoO processor on an FPGA for the purpose of accelerating processor simulations. To the best of our knowledge this is the first work that studies how the area, frequency and most importantly performance of CAM-based instruction schedulers scale with the number of scheduler entries on an FPGA.

## VI. CONCLUSION

In this work we explored part of the design space of instruction schedulers for out-of-order soft processors. We examined the effect of scheduler size, instruction selection policy, and back-to-back scheduling on performance, area and frequency. We showed that in isolation (no restrictions on the clock frequency), a two-entry scheduler with a location-based selection policy and no back-to-back scheduling achieves maximum performance. However, by limiting the processor frequency to 303MHz (the frequency that an FPGA-friendly register renamer operates at) we showed that a four-entry scheduler with age-based selection policy and back-to-back scheduling reaches the maximum performance. The results of this work can be used to estimate the best scheduler design under various operating frequency assumptions. This work is predicated on the assumption that the workloads for future embedded and FPGA-systems will evolve to include workloads with irregular data and control patterns.

## REFERENCES

- [1] J. E. Smith and G. Sohi, "The Microarchitecture of Superscalar Processors," *Proceedings of the IEEE*, 1995.
- [2] P. Yiannacouras, J. G. Steffan, and J. Rose, "VESPA: portable, scalable, and flexible FPGA-based vector processors," in *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2008, pp. 61–70.
- [3] K. Aasaraai and A. Moshovos, "Towards a viable out-of-order soft core: Copy-free, checkpointed register renaming," in *19th International Conference on Field Programmable Logic and Applications (FPL)*, Prague, Czech Republic, September 2009.
- [4] S. Palacharla and J. E. Smith, "Complexity-effective superscalar processors," in *In Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997, pp. 206–218.
- [5] J. Keller, "The alpha 21264 microprocessor architecture," in *In Proceedings of 9th Annual Microprocessor Forum*, 1996.
- [6] "Arcturus Networks Inc., uClinux," <http://www.uclinux.org/>.
- [7] Standard Performance Evaluation Corporation, "SPEC CPU 2006," <http://www.spec.org/cpu2006/>.
- [8] Altera DE3 Development System with Stratix III FPGA, "TERASIC Inc." <http://university.altera.com/materials/boards/de3/>.
- [9] Altera Corp., "Nios II Processor Reference Handbook v9.0," 2009.
- [10] F. J. Mesa-Martínez, M. C. Huang, and J. Renau, "Seed: scalable, efficient enforcement of dependences," in *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. New York, NY, USA: ACM, 2006, pp. 254–264.
- [11] S. Fytraki and D. Pnevmatikatos, "RESIM: A trace-driven, reconfigurable ILP processor simulator," in *Design and Automation Europe*, 2008.
- [12] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *MICRO 40: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 249–261.