

On the Latency, Energy and Area of Checkpointed, Superscalar Register Alias Tables

Elham Safi, Patrick Akl, Andreas Moshovos,
Andreas Veneris

Department of Electrical and Computer Engineering
University of Toronto

{elham, pakl, moshovos, veneris}@eecg.utoronto.ca

Aggeliki Arapoyianni

Department of Informatics
University of Athens

arapoyianni@di.uoa.gr

ABSTRACT

We present two full-custom implementations of the Register Alias Table (RAT) for a 4-way superscalar dynamically-scheduled processor in a commercial 130nm CMOS technology. The implementations differ in the way they organize the embedded global checkpoints (GCs) which support speculative execution. In the first implementation, representative of early designs, the GCs are organized as shift registers. In the second implementation, representative of more recent proposals, the GCs are organized as random access buffers. We measure the impact of increasing the number of GCs on the latency, energy, and area of the RAT. The results support the importance of recent techniques that reduce the number of GCs while maintaining performance.

Categories and Subject Descriptors

B.7.1 Integrated Circuits , C.1 Processor Architecture.

General Terms

Measurement, Performance, Design, Experimentation.

Keywords

Checkpointing, register renaming, energy, latency.

1. INTRODUCTION

In modern processors, register renaming eliminates false data dependencies and hence increases instruction level parallelism (ILP). The Register Alias Table (RAT) implements register renaming by maintaining mappings between the register names utilized by the program (architectural registers) and the physical storage in which the corresponding values reside at any given point in time (physical registers). Register renaming is complicated by control flow speculation, another performance enhancing technique. In control flow speculation, the processor speculates on the direction of a branch and speculatively executes instructions down that path. In case of mispeculation, the processor must recover by reversing the effects of all

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISLPED'07, August 27–29, 2007, Portland, Oregon.

Copyright 2007 ACM 978-1-59593-709-4/07/0008...\$5.00.

mispeculated instructions including all changes to the RAT. Modern processors implement two such recovery methods. The first is the re-order buffer (ROB), which keeps a log of all changes and allows recovery at any instruction, including branches. Using the ROB, recovery time is proportional to the number of mispeculated instructions. As mispeculations are relatively frequent, processors incorporate another mechanism with fixed latency recovery time that uses global checkpoints (GCs). A GC takes a complete snapshot of all relevant processor state, including the RAT, taken upon renaming a branch.

Ideally, it would be possible to allocate a GC to every speculated branch as earlier implementations did [11]. This policy appears to be impractical and inefficient for modern processors that attempt to extract ILP over larger portions of code. Previous work has demonstrated that for modern processors more than 24-48 GCs would be required to avoid a degradation in performance [1][7]. As current checkpointed RAT implementations embed GCs in the RAT, increasing the number of GCs increases RAT latency, energy and power. Motivated by this observation, recent works have proposed methods for reducing the number of GCs while maintaining performance. However, to the best of our knowledge, no previous work quantified the magnitude of RAT latency and energy as a function of the number of GCs. An understanding of the underlying trends is essential in determining the true benefits of previous proposals and justifying the development of further optimizations. Accordingly, this work makes the following contributions: (i) It presents two full-custom implementations of a superscalar RAT with embedded GCs in a commercial 130nm CMOS technology. The first implementation, representative of early RAT designs, organizes the GCs as small bi-directional shift registers next to each RAT bit. The second implementation, representative of recent proposals, organizes the GCs as small random access buffers beside each RAT bit. (ii) It quantifies the magnitude of the latency, energy and area of the RAT as a function of the number of GCs.

Our results quantitatively validate that increasing the number of GCs significantly impacts RAT latency and energy. Moreover, our results serve as motivation for trying to reduce the number of GCs even further.

The rest of this paper is organized as follows. Section 2 reviews RAT checkpointing. Section 3 discusses the two checkpointing implementations and their transistor-level design. Section 4 reviews related work. Section 5 discusses the physical level simulation results. Section 6 concludes this work.

2. CHECKPOINTED RAT

In this work, we restrict our attention to the SRAM-based implementation of a RAT where the RAT is a table which is indexed by an architectural register name and each entry contains the corresponding physical register name [11]. Given a MIPS-like load/store instruction set architecture with up to two source operands and one destination, to rename N instructions per cycle, a RAT must have $3xN$ read and N write ports. $2xN$ read ports are needed to rename the two source operands, and another N read ports are required to read the current mappings for the destination operands for ROB recovery[7]. Finally, N write ports are needed to write the new mappings of the destination registers.

Conceptually, a checkpointed RAT comprises multiple copies of the RAT (Figure 1(a)). A GC is taken by copying the main RAT content into one of the copies. Recovery is done by copying one of GCs to the main RAT. In existing implementations, the GCs are physically next to each main RAT bit (Figure 1(b)) [8][11].

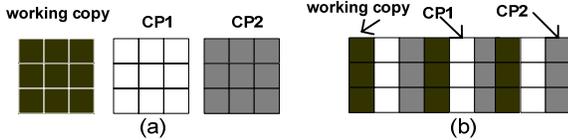


Figure 1: RAT Checkpointing. (a) Conceptual organization. (b) Actual implementation.

3. RAT IMPLEMENTATIONS

Previous work on RAT checkpointing assumes GCs are either organized as a bi-directional shift register or as a random access buffer next to each RAT bit. We will refer to the former as SAB (serial access buffer) and to the latter as RAB (random access buffer). Figures 2(a) and 2(b) depict the RAT cells for SAB and RAB, respectively. The RAT bit (M) has $3xN$ read and N write ports which are not depicted. The designs differ in the way they implement GC allocation and restoration. GC *allocation* takes a snapshot of the current RAT contents, while GC *restoration* restores the RAT from one of the GCs. The GC cells shown in Figure 2 are marked as C_i . In SAB, GC allocation is done by shifting the C_i bits to the right, copying the RAT bit value to its adjacent vacant position. Restoring from a GC may require multiple steps. For example, restoring from C_3 requires three left shifts. In RAB, the GCs are organized as random access buffers; hence, disregarding interconnect delay, GC allocation and restoration latency is the same for all GCs. Both designs require an external controller to keep track of the number of available GCs and to coordinate the GC operations. For SAB, we only need to keep track of the number of allocated GCs. For RAB, we need to track the validity of individual GCs.

3.1 Transistor Level Design

The RAT is a multi-ported register file with embedded GCs. It consists of the precharge and equalization circuitry, sense amplifiers, write drivers, control circuitry, decoders, along with an array of RAT cells connected by bitlines and wordlines.

Figure 3(a) depicts the main RAT cell comprising two back-to-back inverters and several read and write ports. Figure 3(b) shows a complete RAT cell with 16 GCs. Each GC requires an SRAM cell. The GC cells are shown in Figures 3(c) and 3(d) for SAB and RAB, respectively. The multi-ported RAT cell uses one wordline

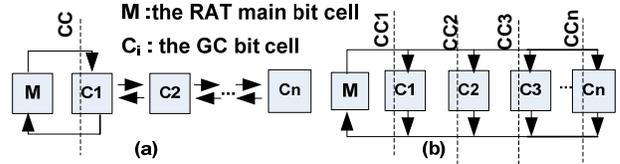


Figure 2: Checkpointed RAT bit. (a) Sequential-Access-Buffer (SAB). (b) Random-Access Buffer (RAB).

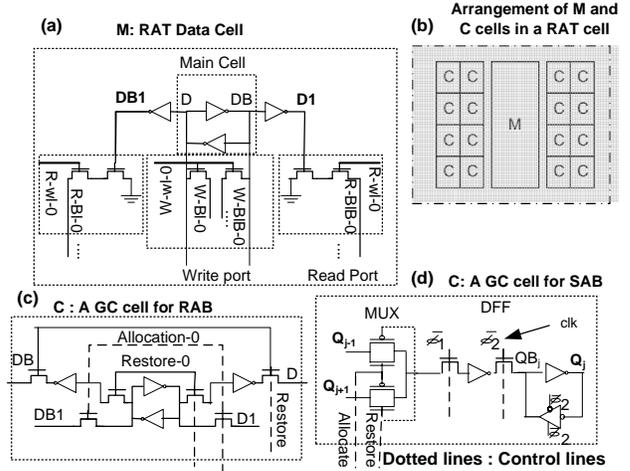


Figure 3: (a) RAT cell with no GC, (b) the layout of the main RAT bit (M) and the GCs (C), (c) RAB GC, and (d) SAB GC.

and two bit lines per write or read port. As multiple reads may access the same RAT entry, and because all GCs are connected via pass gates to the main cell, the main cell should be capable of driving a capacitance proportional to the number of ports and GCs. To protect the data stored in the main cell during multiple accesses, a decoupling buffers isolate the RAT main cell and the read ports [4][12]. The buffers also isolate the GCs. Because of these buffers, separate write bitlines are required. We use differential read and write operations because of their better energy, latency, and robust noise margins. To reduce power, we utilize several techniques: pulse operation for the wordlines, the periphery circuits, and the sense amplifiers; multi-stage static CMOS decoding; and current-mode read and write operations.

Figure 3(c) and (d) show the RAB and SAB GC cells, respectively. In SAB, GCs are organized as bi-directional shift registers with connections between adjacent cells. Only one of the GCs is connected to the main RAT bit through pass gates. A GC cell consists of a register and a multiplexer that controls the direction of the shift. The shift register uses two non-overlapping clocks. SAB requires two external control signals irrespective of the number of GCs. In RAB, every GC cell is connected to the RAT bit cell via pass transistors. Two pairs of pass transistors are used to copy the value from the RAT bit into the GC and vice versa. Each RAB GC cell requires two external control signals.

4. RELATED WORK

Previous works investigated the latency and/or the energy of specific RAT implementations. Bishop et al., present an implementation of a RAT in a 350nm technology for single-issue processors and measure its worst case delay [2]. De Gloria et al.,

present a 4-way superscalar RAT with four GC and embedded cross-bundle dependence detection logic in a 350nm technology and reports its latency [3]. Several works proposed to reduce the number of RAT accesses and the number of RAT ports requirement in order to reduce RAT energy and delay [6][9]. This work complements these studies as it quantitatively investigates RAT energy, delay and area as a function of the number of GCs.

5. EVALUATION

We implemented full-custom layouts for both designs using the Cadence® tool set in a commercial 130nm fabrication technology with a 1.3V supply voltage. We started with minimum size transistors and then increased dimensions to achieve worst case delay that was less than an estimated delay of 600ps for a RAT with no GCs. We arrived at this upper bound by first estimating the delay of a 64-bit, 64-entry SRAM with 12 read and four write ports using CACTI 4.2 [10]. We further corroborated this estimate by using linear technology scaling on previously published estimations for multi-ported register files [5][9]. Speed can be improved further at the expense of energy. We used the Spectre™ simulator for circuit simulation and we report worst case delay and energy. Due to lack of space, we limit our attention to the RAT of a 4-way dynamically scheduled superscalar processor. We assume 64 and 512 architectural and physical registers, respectively. The base RAT has 12 read ports, four write ports, no GCs, and 64 entries of nine bits.

5.1 Delay

Figure 4(a) shows the read latency as a function of the number of GCs. A read progresses through the address latch, the decoder, the wordline driver, the bitlines and the sense amplifier. Increasing the number of GCs elongates the wordlines and the bitlines as we attempt to keep the geometry of each RAT cell as square as possible. The read latency of RAB is higher than that of SAB as the main RAT bit is connected via pass transistors to each of the GC bits. Resizing the RAT bit cell, the pass transistors and the bitline drivers avoids a significant increase in latency. However, this further elongates the wordlines and bitlines and hence after a point it becomes ineffective. The read delay with RAB increases from 597ps with four GCs to 646ps with 16 GCs, an 8.3% increase. With SAB, the delay increases from 595ps to 621ps as going from four to 16 GCs, a 4.4% increase.

Figure 4(b) shows the write latency as a function of the number of GCs. For SAB, increasing the GCs from four to 16 increases the latency from 546ps to 608ps, a 11.4 % increase. For RAB, write latency is more sensitive than read latency. RAB has additional pass transistors per GC bit and hence requires a larger RAT cell. For RAB, latency increases from 551ps to 672ps, a 22% increase, going from four GCs to 16.

Figure 5 shows the latency for GC allocation and restoration. While these latencies increase with the number of GCs, in absolute terms the latencies are much less than those of reads and writes. Although GC allocation and restoration latencies for SAB are less than those of RAB, completely restoring the RAT under SAB may require multiple shifts as addressed in Section 3. For SAB, the reported GC allocation delay and GC restoration delay are respectively the delay of copying the nearest GC to the RAT and the delay of copying the RAT bit to the nearest GC.

During each cycle, instructions need to first read from the RAT and then write the new mappings to it. In the simplest

implementation, the read and write phases are serialized within a single cycle. Figure 6 shows the maximum operating frequency for the RAT under this assumption. We ignore pipelining overheads and the possibility of pipelining the RAT itself. Nevertheless, this models a realistic implementation. For SAB, the operating frequency drops by 1.1%, 3% and 8.2% with four, eight and 16 GCs, respectively. The reference is the RAT with no GCs. For RAB, corresponding measurements are 1.7%, 5% and 14.4%. The performance of RAB deteriorates more rapidly than that of SAB. With eight and 16 GCs, SAB is 2.1% and 7.3% faster than RAB.

5.2 Energy

Figures 7(a) and 7(b) depicts the worst case energy for RAT reads and writes. We measure energy assuming that all 12 read and all four write ports are active. Both read and write energy increase with the number of GCs. RAB requires more energy than SAB. Read energy for RAB increases by 4.6% when the number of GCs increases from four to eight, and by another 13.2% going to 16. The corresponding measurements for SAB are 2.1% and 7.4%. Write energy with RAB increases by 15.6% as the number of GCs increases from four to eight, and by another 48.8% when the number of GCs increases to 16. The corresponding measurements for SAB are 5.9% and 19.4%, respectively. However, write energy is at most about one third of RAT read energy.

Figures 8(a) and 8(b) show the worst case energy for GC allocation and restoration operations as a function of the number of GCs. For both operations, energy increases linearly with the number of GCs for SAB; these operations are implemented as shifts in the GC queue and all GC cells participate. GC allocation energy for SAB increases by 81% going from four GCs to eight, and by another 93% going from eight to 16 GCs. RAB behaves differently. GC allocation is implemented as a random access write to a single GC cell. Accordingly, we expect write energy to increase only slightly with the number of GCs. GC allocation energy for RAB increases by 10% going from four GCs to eight, and by another 32.1% going from eight to 16 GCs. GC restoration exhibits similar behavior but in absolute terms it requires more energy than GC allocation. The RAT bit cell includes additional drivers for the read and write ports. Hence, it has a much higher capacitive load and changing its value requires much more energy compared to changing the value of a GC bit. GC restoration energy for RAB increases by 4.2% going from four to eight, and by another 18.3% going from eight to 16. For SAB, these measurements are 101% and 105%, respectively.

5.3 Area

Figure 9 reports RAT area as a function of the number of GCs. SAB is smaller than RAB since it uses fewer control signals and more localized connections. Increasing the number of GCs from four to eight increases RAT area by 10.6% and 10.2% for RAB and SAB, respectively. Area increases by 13.1% and 12.5% going from eight to 16 for RAB and SAB, respectively.

5.4 Summary

The results demonstrate that having more than four GCs impacts RAT latency considerably for both RAT implementations, more so for RAB. With 16 GCs, SAB and RAB are 8.2% and 14.4% slower than the base RAT with no GCs, respectively. GC allocation and restoration are much faster than RAT reads and writes. Embedding 16 GCs increases energy per operation by

22.3% and 11.3% for RAT read, and by 91.3% and 31.4% for RAT write for RAB and SAB, respectively. GC allocation and restoration energy increases with the number of GCs almost linearly for SAB. With 16 GCs, RAB and SAB require 40% and 31.4% more area the base RAT with no GCs, respectively.

The results highlight the importance of previous work that aimed at reducing the number of GCs to four or fewer. They also demonstrate that focusing exclusively on instructions per cycle (IPC) can be misleading. RAB increases overall RAT latency compared to SAB. If this increase impacts the critical path, then it would lead to a proportionate decrease in performance which will offset some of the IPC benefits of methods that rely on RAB.

6. CONCLUSIONS

In this work, we have investigated the latency, energy and area of a superscalar RAT as a function of the number of embedded GCs. We studied two full-custom implementations for a checkpointed RAT in a 130nm technology. Our results demonstrate quantitatively that increasing the number of GCs drastically increases latency, energy, and area of the RAT. The results justify GC optimizations that reduce the number of GCs. Moreover, this work serves as motivation for developing checkpoint/restore techniques requiring very few (four or less) GCs.

7. ACKNOWLEDGMENTS

This work was supported by an NSERC Discovery grant and funds from the University of Toronto. This work was also supported by the European Union - European Social Fund & National Resources - EPEAEK II.

8. REFERENCES

- [1] H. Akkary, et al., "An Analysis of Resource Efficient Checkpoint Architecture", ACM Transaction on Architecture and Code Optimization, 1(4): 418-444, Dec. 2004.
- [2] B. Bishop, et al., "The Design of a Register Renaming unit", Great Lakes Symposium on VLSI, Mar. 1999
- [3] A. De Gloria, et al., "An Application Specific Multi-Port RAM Cell Circuit for Register Renaming Units in High Speed Microprocessors", IEEE International Symposium on Circuits and Systems, 4:934 - 937, May 2001.
- [4] R. Heald et al., "A Third-Generation SPARC V9 64-bit Microprocessor", IEEE Journal of Solid-State Circuits, 35(11) : 1526-1538, Nov. 2000.
- [5] R.K. Krishnamurthy, et al., "130-nm 6-GHz 256 × 32 bit Leakage-Tolerant Register File", IEEE Journal of Solid-State Circuits, 37(5): 624-632, May 2002
- [6] G. Kuçuk, et al., "Reducing Power Dissipation of Register Alias Tables in High-Performance Processors, IEE Proceedings on Computers and Digital Techniques, 152(6): 739 - 746, Nov. 2005.
- [7] A. Moshovos, "Checkpointing Alternatives for High Performance, Power-Aware Processors", IEEE International Symposium on Low Power Electronic and Design, 318 -321, Aug. 2003
- [8] S. Palacharla, "Complexity-effective Superscalar Processors", Ph.D. Thesis, University of Wisconsin-Madison, 1998.
- [9] R. Sangireddy, "Reducing Rename Logic Complexity for High-Speed and Low-Power Front-End Architectures", IEEE Transactions of Computers, 55(6):672- 685, Jun. 2006.
- [10] D. Tarjan, S. Thoziyoor and N. P. Jouppi, *CACTI 4.0*, HP Labs Technical Report HPL-2006-86, 2006.
- [11] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor", IEEE MICRO, 1996.
- [12] V. Zyuban, "Inherently Lower-Power High-Performance Superscalar Architectures", PhD Thesis, University of Notre Dame, Jan. 2000.

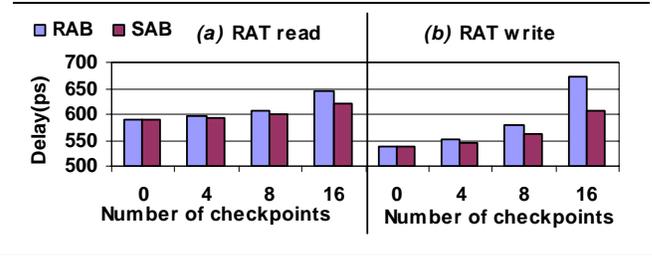


Figure 4: RAT read and write delay.

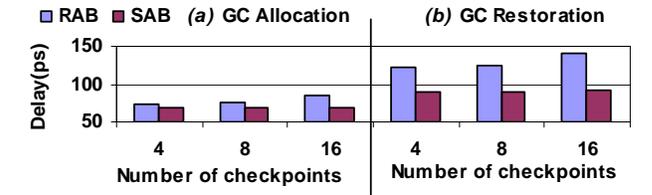


Figure 5: GC allocation and restoration delay.

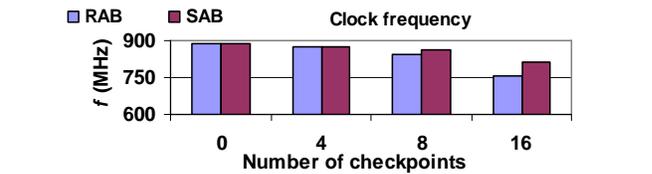


Figure 6: Maximum RAT operating frequency.

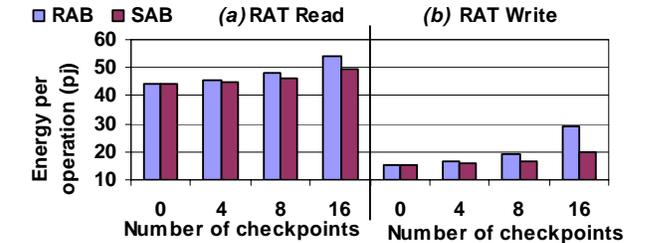


Figure 7 : RAT energy per read and write operation.

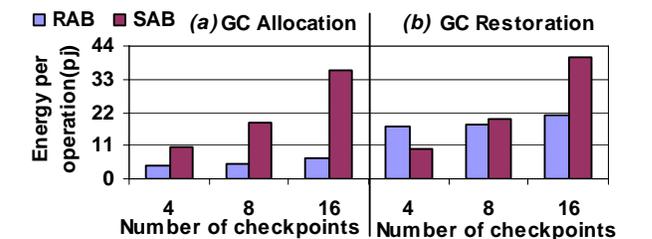


Figure 8 : RAT energy for GC allocation and restoration.

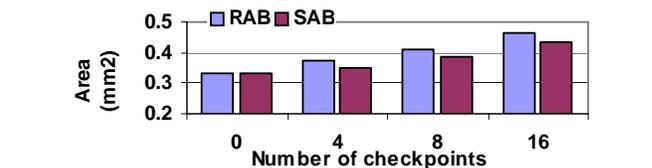


Figure 9 : RAT area.