Reducing Register File Power Consumption by Exploiting Value Lifetime Characteristics

Zhigang Hu and Margaret Martonosi
Department of Electrical Engineering
Princeton University
{hzg,mrm}@ee.princeton.edu

Abstract

With the trend towards wider instruction issue and larger instruction windows, register files grow both in terms of size and number of read/write ports. However, large multi-ported register files consume a substantial amount of power, and may also limit the cycle time of a processor. This work attempts to address these issues by taking advantage of the facts that many register accesses show extremely good temporal locality, and that many register values are very short-lived.

Based on these ideas, this paper proposes buffering the result between functional units and the register file in a "holding tank" we call the Value Aging Buffer (VAB). Due to the temporal locality of value access, most of the accesses to register values can be serviced from the VAB. Since the VAB is smaller than a typical register file, it can have better power characteristics. Furthermore, reducing read/write requests at the register file also allows us to reduce the number of ports provided on the register file. Thus, this gives us multiple ways in which register file power consumption is reduced. For example, with a 16-entry VAB, register file read and write operations can be reduced by 64% and 61% respectively for SPECint95 benchmarks. The VAB offers roughly a 30% power savings for register files, with less than a 5% performance loss.

1 Introduction

Current microprocessor design has a tendency towards wider instruction issue and larger instruction windows. This trend also naturally leads to a larger register file with more read/write ports. In early work, Johnson suggested that for a 4-issue processor, register file read ports could be reduced from 8 to 4 with only a minor loss of performance, if all registers can be accessed within one cycle [6]. However, reducing register file ports requires arbitration logic that is complex and slow and thus may stretch the cycle time.

In [3] the authors showed that 80 and 120 physical registers are needed respectively for a 4-issue and 8-issue superscalar processor. To manage the cycle time pressure inherent in such a design, they proposed a decentralized register file organization. Such clustered notions have been embodied in current commercial processors such as the Alpha 21264 [5].

Previous work has shown that register value access shows strong temporal locality [4]. In fact, most of the read and write operations of a register occurs within a few cycles after the value is first produced. Based on this observation, the authors in [4] proposed using local register files within each processing unit in Multiscalar processors to reduce the traffic of the global, ISA-visible register files. Significant traffic reduction can be achieved by using such local register files.

In this paper, we similarly leverage value lifetime characteristics. Our primary goal, however, is to quantify the power reduction that can be accomplished by taking advantage of short-lived variables. We first give an analysis of value lifetime characteristics based on our simulation of a model processor similar to current

state-of-art superscalar processors. We show that after a physical register is allocated, it typically waits an average of 12 cycles before it actually holds valid data. Within only the next 3-4 cycles, however, the data is typically finished being used. It will then stay idle for about 8 cycles to maintain the machine state for precise exceptions.

Since most of data accesses are clustered within a few cycles after data is produced, we propose a small structure, which we call the Value Aging Buffer (VAB), which sits between the functional units and the register file. The VAB is intended to buffer recently-produced results for several cycles in order to "filter" accesses that would otherwise go to the larger and more power-hungry register file. Since the buffer can be much smaller than the register file, the total power consumption can be reduced significantly. In our simulations, we observe a power saving of 30% with a performance loss of less than 5%.

In fact, the Value Aging Buffer also seems to improve the cycle time scenario for superscalar processors. Most of the data accesses can be serviced from the smaller, presumably-faster VAB. Accesses to the register file are backgrounded, similar to L2 cache accesses, and thus the register file is no longer on the critical path determining CPU cycle time. For this reason, we feel that the VAB may degrade performance by even less than 5% in many cases.

1.1 Structure of this paper

In Section 2, we explain the simulation environment and machine model used to evaluate our ideas. Next in Section 3, we present statistics on value lifetime patterns; these motivate our proposal for the Value Aging Buffer. Section 4 discusses the detailed structure and operations of the Value Aging Buffer and shows its power and performance result compared to a conventional 8-read-port and 4-write-port register file. Finally, Section 5 concludes the paper and discusses our plans for future work.

2 Simulation Framework

2.1 Simulator

Our model processor has sizing parameters that closely resemble Alpha 21264 [5] with the key difference that we do not model a clustered organization. The processor parameters are shown in Table 1. Our power consumption results are derived from Wattch [1] which is an architecture-level power modeling framework based on Simplescalar [2]. Wattch uses a suite of parameterizable power models for different hardware structures and per-cycle resource usage counts generated through cycle-level simulation. Unlike the original Wattch and Simplescalar, this paper's simulator models a physical register file rather than an RUU. This aids our value lifetime analysis and makes our power analysis for the physical register file more accurate.

Parameter	Value
Processor Core	
Physical Registers	64-INT, 64-FP
Fetch width	4 instructions per cycle
Decode width	4 instructions per cycle
Issue width	4 instructions per cycle
Commit width	4 instructions per cycle
Functional Units	4 IntALU, 1-IntMult/Div, 2 FP ALU, 1-FPMult/Div 2 MemPorts
Branch Prediction	
Branch Predictor	Combined, Bimodal 4K table, 2-Level 1K table 10bit history, 4K chooser
BTB	1024-entry, 2-way
Return Address Stack	32-entry
Mispredict penalty	7 cycles
Memory Hierarchy	
L1 Dcache Size	64K, 2-way, 32B blocks
L1 Icache Size	64K, 2-way, 32B blocks
L2	Unified, 2M, 4-way LRU, 32B blocks,12-cycle latency
Memory	100 cycles
TLB Size	128-entry, 30-cycle miss penalty

Table 1: Configuration of Processors

2.2 Benchmark

We choose to evaluate the power and performance property of Value Aging Buffer by simulating the SPEC95 Integer benchmark suites. We compiled the benchmarks for the Alpha instruction set using the Compaq Alpha cc compiler with the following optimization options as specified by the SPEC Makefile: -migrate –std1 –O5 –ifo –non-shared. For each program, we simulated 50M

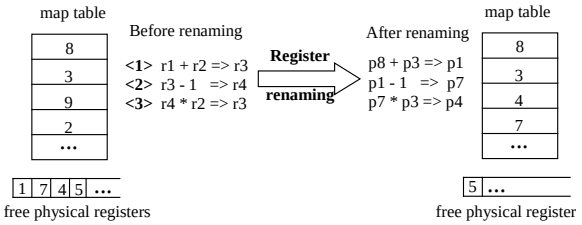

Figure 1: Register renaming mechanism

instructions using the ref input set.

3 Value Lifetime Analysis

3.1 Background and Architectural Model

In our model processor, the instruction set has 32 architectural registers. A map table records the mapping between these virtual architectural registers and 64 physical registers. This mapping occurs during decoding and a new physical register is assigned if the corresponding instruction produces a value. This new physical register is filled with data when the instruction completed execution in the writeback stage. Subsequent instructions can fetch the data either from the result buses if applicable or read from the physical register file. The value is kept after its last use in case it is needed to recover from a mispredicted branch or an exception. It can be safely freed only after a new instruction with the same destination architectural register number is committed without error, or when the corresponding instruction is squashed due to misprediction or exception. Figure 1 show an example of the register renaming mechanism.

3.2 Value Lifetime Characteristics

The life of a physical register value can be divided into 3 periods as shown in Figure 2.

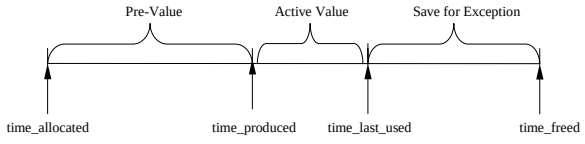

Figure 2: 3 periods in the life cycle of a register value

- Pre-value: This is the time period between when the register tag has been allocated (time_allocated) and when the value is produced (time_produced). There is no useful data so no reads or writes will occur in this period.

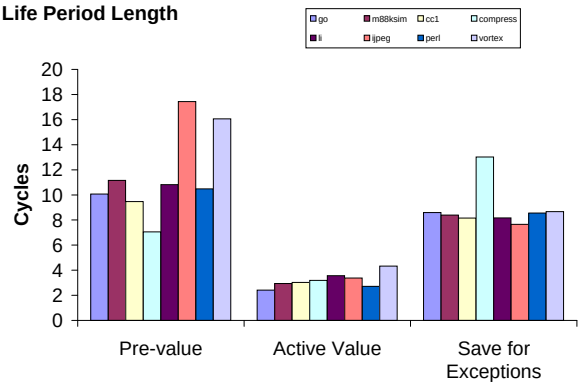

Figure 3: Length of the 3 periods in the life cycle of a register value

- Active Value: This is the time period between when the value is first produced and put in the register, until the last read or write that uses the value (time_last_used). All read and write operations occur in this period.

- Save for exceptions: This time period follows the last planned read or write to the value. During this time period, the register value is maintained for the sole purpose of guaranteeing precise exceptions. The value will only be used if an exception occurs in the window after time_last_used and before time_freed.

Figure 3 shows the average length of the 3 periods for SPECINT 95 benchmarks. From the figure we can see that on average the "Active Value" period of a value is only 3-4 cycles. This indicates that all the read and write operations are typically clustered within a few cycles after it is produced.

4 Value Aging Buffer

The previous section illustrated the fact that values have quite short active lifetimes on average. Frequently, values have seen their last use before they are even committed to an architectural register. Prior work has suggested register files that take advantage of short-lived variables [10] in order to improve cycle times. The primary goal of our work is to suggest an approach whose specific goal is to harness short-lived variables in order

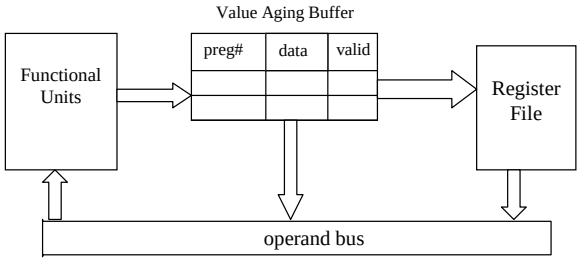

Figure 4: Structure of the Value Aging Buffer

to reduce power consumption with little performance impact.

In particular, we propose a *value aging buffer* that sits between the functional units and the register file as illustrated in Figure 4. The value aging buffer (VAB) acts as a holding tank. Our goal is to postpone (typically-expensive) accesses to the register file long enough to see if they are actually necessary. If they are not necessary, then we can often avoid the power-hungry step of writing to the large multi-ported register file. This approach is philosophically similar to using a small filter cache [7] in front of the data cache to save power.

4.1 Read Operations

When a functional unit requires an operand value, it first performs a lookup in the VAB. If it is found there, it can be retrieved within one cycle. If it is not found in the VAB, the request is forwarded to the register file and the value can be fetched with a one-cycle penalty. This is the approach evaluated here, but it could be refined to allow for locating values with a fixed access time. We define the VAB miss rate as the number of misses in the VAB divided by the total number of read request to the VAB. A low VAB miss rate means better power and performance characteristics.

4.2 Write Operations

When a functional unit produces a value, it is first written to the VAB. The VAB acts as a FIFO. Once full, it will evict the oldest entry each time a new entry needs to be added. We define the VAB eviction rate as the ratio of VAB entries evicted to the register file. A low eviction rate means fewer register file writes and thus will lead to power savings. Since the VAB receives results from functional units, it can have a one-cycle warning of when new results will be produced. This allows it to begin these evictions in advance.

4.3 Freeing Operations

When a physical register is freed, the corresponding entry in the VAB will also be freed, if one exists. A physical register can be freed either when the producer instruction is squashed due to misprediction or exception or when another instruction with the same destination register is commited. In either case, the physical register number is sent to the VAB, compared with the physical register number in each entry, and the matched entry is freed by resetting their valid bit. Freeing register values in the VAB reduces the write traffic to the physical register file and thus reduces the power consumption of the physical register file.

4.4 Exception and Misprediction Handling

In our model processor, precise exceptions are maintained. When exceptions occur, the content of VAB is dumped into the register file. Since exceptions are relatively rare, the impact on performance and power is small. For misprediction, the handling should be fast since this will add to the misprediction penalty. In our model processor, misprediction is handled with a branch stack [12]. Similarly, we can put the VAB in the branch stack so that when misprediction is detected, the content of VAB can be copied back quickly.

Since the VAB is small and we have an additional cycle to access the physical register file, we assume we have enough cycle time budget to add an arbiter before the VAB and the physical register file so that their read ports can be reduced. In our simulation, we assume the VAB and the physical register file have 5 and 3 read ports respectively. They both have 4 write ports to avoid an arbiter for write.

4.5 Performance and Power Results

Figure 5 shows the miss rates for VAB of size 8, 12 and 16 respectively. The main observation is that a relatively small VAB of 16 entries is sufficient to capture roughly 50-70% of the read requests. Figure 6 shows the eviction rates for VAB of size 8, 12 and 16. With a

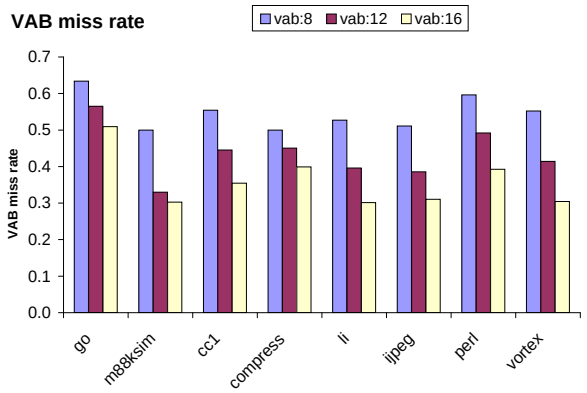

Figure 5: Ratio of reads serviced from the register file

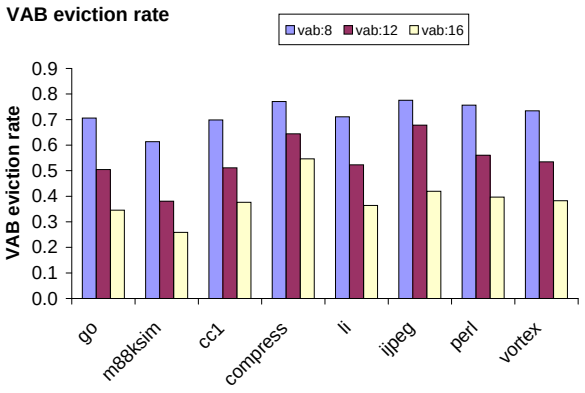

Figure 6: Ratio of writes to the register file

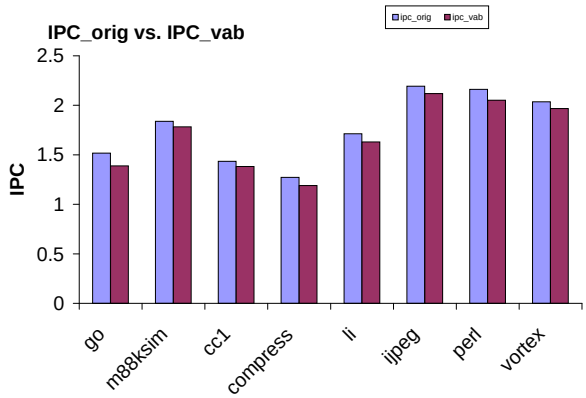

Figure 7: IPC with VAB vs. without VAB

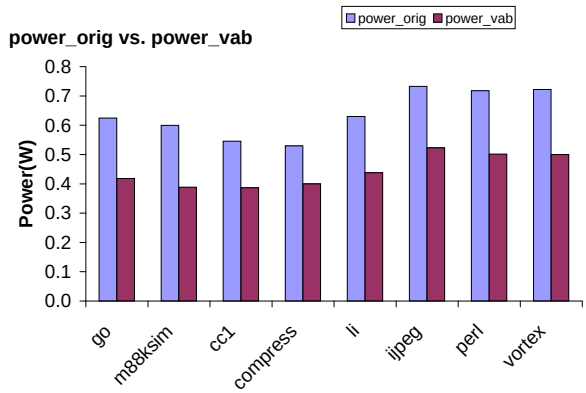

Figure 8: Power of VAB + register file with VAB vs. Power of register file without VAB

16-entry value aging buffer, only 26%-56% of all value produced need to actually save to the register file. All other values will be freed during their stay in the VAB.

VABs smaller than 16 entries are not as effective. This is because small VABs require faster aging or forced evictions to the register file. These events increase the amount of register file activity, and reduce the benefit of the VAB as a filter in front of the register file.

Our main goal in implementing a VAB is to reduce power consumption without unduly hurting performance. Thus, Figures 7 and 8 show the IPC and power consumption of our proposed approach. Note that for the VAB scheme we show the total power consumption of both the VAB and the register file. Compared to a register file without VAB, a 16-entry VAB incurs an IPC loss of less than 5%, but saves about 30% of the register file power.

4.6 Value Aging Buffer with location bit

In our original VAB scheme, we assume it will take 2 cycles to fetch an operand if it is in the register file. This is the major source of the performance loss. To eliminate this one-cycle penalty, we can associate an additional bit with each physical register to indicate whether it is in Value Aging Buffer or physical register file. Whenever a value is needed, the corresponding bit is first checked to decide whether the request should go to the VAB or the physical register file. Since the additional work is only a one-bit check, we assume it does not lead to an increase of the cycle time compared with the original 8-read-port/4-write-port scheme. Thus in this case, the performance will not experience a loss since all data can be fetched within a cycle, whether it is in the VAB or the physical register file. However, since a large portion of the accesses can be serviced from the smaller VAB, we can still expect significant power savings.

In this simulation we assume the physical register file has 8 read ports to avoid an arbiter since all physical registers should be able to be accessed within one cycle. Compared to the original VAB scheme, the physical register has more read ports so the power consumption

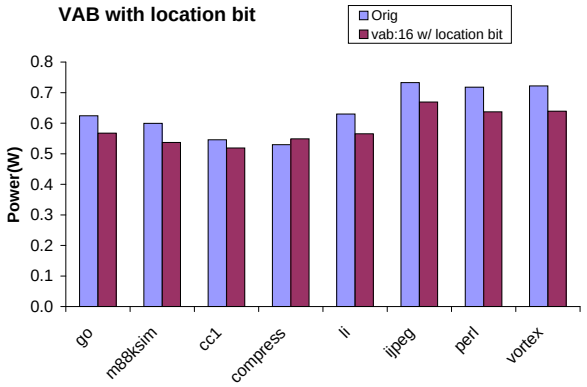

Figure 9: Power of VAB + register file with VAB with location bit vs. Power of register file without VAB

is larger. This leads to a smaller power saving. Figure 9 shows the average power consumption when using a 16-entry VAB with a location bit for each physical register. An average power saving of 9% is achieved without impacting the performance. Though the power savings is smaller than with the original scheme, the additional bit helps with both performance and fast exception handling which makes this scheme a feasible alternative.

5 Conclusions and Future Work

This work introduces ideas for register file power reductions based on taking advantage of typical register access characteristics. These ideas will be broadened and expanded upon in our future work. For example, we intend to look for further ways to exploit short-lived variables. Dynamic approaches we have examined thus far indicate roughly 30-65% of values are short-lived, meaning that their destination register will be over-written by an unspeculative instruction already in the instruction window. We intend to explore further dynamic techniques and also expand on previously-proposed compiler techniques [8] [9] to increase and detect short-lived variables. We also wish to more aggressively utilize information on the value access patterns to reduce processor power consumption.

More broadly, micro-architectures are progressing rapidly towards several styles of post-superscalar architectures with higher performance and manageable power consumption. In further work, we intend to explore the performance and power-effectiveness of using VABs in a more diverse set of microarchitectures, such as simultaneous multithreading (SMT) [11], Multiscalar [10] and other architectures.

Overall, this paper first analyzed the access pattern of the physical register file values. We showed that for SPECint95 benchmarks on average all the reads and writes are performed within 3-4 cycles after the value is produced. Based on this result, we proposed the VAB to buffer results between functional units and the physical register file. The smaller VAB can service most of the accesses and thus the CPU power is reduced. We feel that our proposal is a first step towards identifying power-effective techniques to employ in a range of modern micro-architectures.

References

[1] D. Brooks, V. Tivari, and M. Martonosi. Wattch: A Framework for Architecture-Level Power Analysis and Optimizations. ISCA 2000.

[2] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., July 1996.

[3] K. Farkas, N. Jouppi, and P. Chow. Register file design considerations in dynamically scheduled processors. Technical report, Compaq Western Research Lab, 1995.

[4] M. Franklin and G. Sohi. Register traffic analysis for steamlining inter-operation in fine-grain parallel processors. In *25th Annual International Symposium on Microarchitecture*, pages 236–245, December 1992.

[5] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, pages 11–16, Oct. 28, 1996.

[6] M. Johnson. *Superscalar Microprocessor Design*. Prentice-Hall, 1991.

[7] M. G. Johnson Kin and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proc. of the 30th Int'l Symp. on Microarchitecture*, Nov. 1997.

[8] L. A. Lozano and G. R. Gao. Exploiting short-lived variables in superscalar processors. In *Proc. Micro-28*, pages 292–302, Dec. 1995.

[9] M. M. Martin, A. Roth, and C. N. Fischer. Exploiting dead value information. In *Proc. of the 30th Int'l Symp. on Microarchitecture*, Nov. 1997.

[10] G. Sohi, S. Breach, and T. N. Vijaykumar. Multiscalar Processors. In *Proc. of the 22nd Int'l Symp. on Computer Architecture*, pages 414–425, June 1995.

[11] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc. of the 22nd Int'l Symp. on Computer Architecture*, pages 392–403, June 1995.

[12] K. C. Yeager. The MIPS R10000 Superscalar Microprocessor. *IEEE Micro*, 16(2):28–40, Apr. 1996.