RC 22380 (W0203-071) March 22, 2002

IBM Research Report

Tradeoffs in Power-Efficient Issue Queue Design

Alper Buyuktosunoglu, David H. Albonesi
University of Rochester
Dept. of Electrical and Computer Engineering

Pradip Bose, Peter W. Cook, Stanley E. Schuster
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Tradeoffs in Power-Efficient Issue Queue Design

Alper Buyuktosunoglu, David H. Albonesi
University of Rochester
Dept. of Electrical and Computer Engineering

Pradip Bose, Peter W. Cook,
Stanley E. Schuster
IBM T. J. Watson Research Center

ABSTRACT

A major consumer of microprocessor power is the issue queue. Several microprocessors, including the Alpha 21264 and POWER4^{TM}, use a compacting latch-based issue queue design which has the advantage of simplicity of design and verification. The disadvantage of this structure, however, is its high power dissipation.

In this paper, we explore different issue queue power optimization techniques that vary not only in their performance and power characteristics, but in how much they deviate from the baseline implementation. By developing and comparing techniques that build incrementally on the baseline design, as well as those that achieve higher power savings through a more significant redesign effort, we quantify the extra benefit the higher design cost techniques provide over the more straightforward techniques.

1. INTRODUCTION

There are many complex tradeoffs that must be made to achieve the goal of a power-efficient, yet high performance design. The first is the amount of performance that must be traded off for lower power. A second consideration that has received less attention is the amount of redesign and verification effort that must be put in to achieve a given amount of power savings. Time-to-market constraints often dictate that straightforward modifications of existing designs take precedence over radical approaches that require significant redesign and verification efforts. For the latter, there must be a clear and demonstrable power savings with minimal negative consequences to justify the extra effort.

One microprocessor structure that has received considerable attention is the issue queue. The issue queue holds decoded and renamed instructions until they issue out-of-order to appropriate functional units. Several superscalar processors such as the Alpha 21264 [11] and POWER4 [10], implement a *latch-based* issue queue in which each entry consists of a series of latches [1, 4]. The queue is *compacting* in that the outputs of each entry feed-forward to the next entry to enable the filling of "holes" created by instruction issue. New instructions are always added to the tail position of the queue. In this manner, the queue maintains an oldest to youngest program order within the queue. This simplifies the implementation of an oldest-first issue priority scheme. Additional important advantages of this implementation are that it is highly modular and can use scannable latches, which simplifies issue queue design and verification.

However, the high price of this approach is its power consumption: for instance, the integer queue on the Alpha 21264 is the highest power consumer on the chip [11]. Similarly, the issue queue is one of the highest power-density regions within a POWER4-class processor core [1]. For this reason, several techniques for reducing the issue queue power have been proposed [2, 3, 5]. However, these prior efforts have exclusively focused on approaches that require considerable re-design and verification effort as well as design risk. What has been thus far lacking is a quantitative comparison of a range of issue queue power optimization techniques that vary in their design effort/risk, in addition to their power savings and performance cost. Our analysis results in several possible issue queue design choices that are appropriate depending on the redesign and verification effort that the design team can afford to put in to achieve a lower-power design.

2. NON-COMPACTING LATCH-BASED ISSUE QUEUE

Figure 1 illustrates the general principle of a latch-based issue queue design. Each bit of each entry consists of a latch and a multiplexer as well as comparators (not shown in this figure) for the source operand IDs. Each entry feeds-forward to the next queue entry, with the multiplexer used to either hold the current latch contents or load the latch with the contents of the next entry. The design shown in Figure 1 loads dispatched instructions into the uppermost unused queue entries. "Holes" created when instructions issue are filled via a *compaction* operation in which entries are shifted downwards. By dispatching entries into the tail of the queue and compacting the queue on issue, an oldest to youngest program order is maintained in the queue at all times, with the oldest instruction lying in the bottom of the queue shown in Figure 1. Thus, a simple position-based selection mechanism like that described in [9], in which priority moves from "lower" to "upper" entries, can be used to implement an *oldest-first* selection policy in which issue priority is by instruction age. Although compaction operation may be necessary for a simpler selection mechanism, it may be a major source of issue queue power consumption in latch-based designs. Each time an instruction is issued, all entries are shifted down to fill the hole, resulting in all of these latches being clocked. Because lower entries have issue priority over upper entries, instructions often issue from the lower positions, resulting in a large number of shifts and therefore, a large amount of power dissipation.

To eliminate the power-hungry compaction operation, we can make the issue queue *non-compacting* [7, 12]. In a non-compacting queue, holes that result from an instruction issue from a particular entry are not immediately filled. Rather, these holes remain until a new entry is dispatched into the queue. At this point, the holes are filled in priority order from bottom to top. However, in a non-compacting queue the oldest to youngest priority order of the instructions is lost. Thus, the use of a simple position-based selection mechanism like that described in [9] will not give priority to older instructions as in the compacting design. Figure 2 shows a simple example of a four-entry issue queue from which a single instruction can issue each cycle and in which lower instructions have priority over upper ones. Initially, both the compacting and non-compacting queues maintain the oldest to youngest order (A, B, C, D). When instruction B issues from the compacting queue, instructions C and D are shifted down and a new instruction (E) is put into the uppermost location. By contrast, in the non-compacting queue, instruction E is placed in the position formerly occupied by instruction B, giving E higher

priority than instructions C and D. If both of the operands for instruction E are ready when it is inserted into the queue, then in the non-compacting queue it will issue in the next cycle, ahead of the older instruction C. Thus, these older instructions C and D may stay in the queue for a significant period of time, which may have a non-negligible impact on CPI performance.

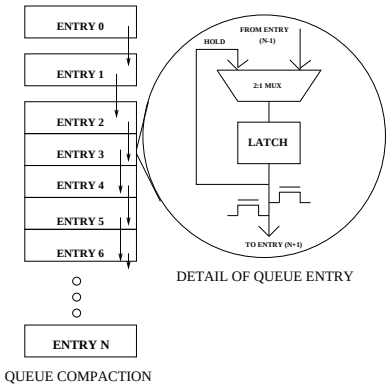

Figure 1: Latch-based issue queue design with compaction.

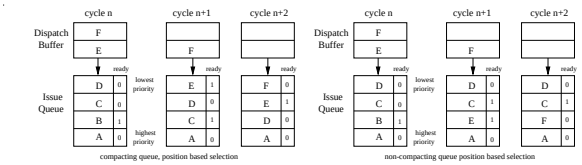

Figure 2: An example showing the differences between compaction and non-compaction with position-based selection.

To solve the problem of lost instruction ordering while maintaining much of the power-efficiency advantages of a non-compacting queue, the reorder buffer (ROB) numbers (sequence numbers) that typically tag each dispatched instruction can be used to identify oldest to youngest order. However, a problem arises with this scheme due to the circular nature of the ROB which may be implemented as a RAM with head and tail pointers. For example, assume for simplicity an 8-entry ROB where the oldest instruction lies in location 111 and the youngest in 000. When an instruction commits, the head pointer of the ROB is decremented to point to the next entry. Similarly, the tail pointer is decremented when an instruction is dispatched. With such an implementation, the oldest instruction may no longer lie in location 111 in our working example, but in any location. In fact, the tail pointer may wrap around back to entry 111 such that newer entries (those nearest to the tail) may occupy a higher-numbered ROB entry than older entries [6]. When this occurs, the oldest-first selection scheme will no longer work properly.

This problem can be solved by adding an extra high-order sequence number bit which we call the *sorting bit* that is kept in the issue queue. As instructions are dispatched, they are allocated a sequence number consisting of their ROB entry number appended to a sorting bit of 0. These sequence numbers are stored with the entry in the issue queue. Whenever the ROB tail pointer wraps around to entry 111 in our example, all sorting bits are flash set to 1 in the issue queue. Newly dispatched instructions, however, including the one assigned to ROB entry 111, continue to receive a sorting bit of 0 in their sequence numbers. These steps, which are summarized in Figure 3, guarantee that these newly dispatched instructions will have a lower sequence number than prior (older) instructions already residing in the queue.

Once the sorting bit adjustment is in place, older instructions can properly be selected from the ready instructions as follows. The most significant bit of the sequence numbers of all ready instructions are ORed together. If the result of the OR is 1, all ready instructions whose most significant bits are 0 will be removed from consideration. In the next step, the second most significant bit of the sequence numbers of all ready instructions that are still under consideration are ORed together. If the result of the OR is 1, all ready instructions still under consideration whose second most significant bits are 0, will be removed from consideration. The Nth step is the same as step 2, except the least significant bit of the sequence number is used. At the end of this step, all ready instructions will have been removed from consideration except for the oldest.

However, this OR-based arbitration mechanism requires a final linear O(N) chain from highest order to lowest order bit. This significantly increases the delay of the selection logic compared to the selection logic described by Palacharla [9], after 4 bits with a 32 entry queue. Note that for a processor that has up to 128 instructions (ROB of 128 entries) in flight, full sequence number consists of 7 bits and a sorting bit. The lack of full age ordering with 4-bit sequence numbers results in a CPI degradation (shown in Section 4.1), although this is an improvement over the CPI degradation incurred with no age ordering (position-based selection with non-compaction).

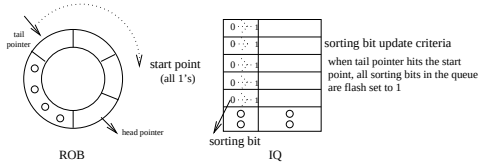

Figure 3: Mechanism for updating the sorting bit in the issue queue.

3. CAM/RAM-BASED ISSUE QUEUES

In this section, we describe issue queue power-saving optimizations that require redesigning the baseline latch-based queue as a CAM/RAM structure in which the source operand numbers are placed in the CAM structure and the remaining instruction information is placed in the RAM structure. The number of entries corresponds to the size of the issue queue. The CAM/RAM structure is arguably more complex in terms of design and verification time and it does not support compaction. However, because of the lower power dissipation of CAM/RAM logic relative to random logic, the CAM/RAM-based issue queue approach has the potential to reduce the average power dissipation of the queue.

While potentially consuming less power than a latch-based solution, a CAM/RAM-based issue queue still offers opportunities for further power reductions. CAM and RAM structures require precharging and discharging internal high capacitance lines and nodes for every operation. The CAM needs to perform tag matching operations every cycle. This involves driving and clearing high capacitance tag-lines, and also precharging and discharging high capacitance matchline nodes every cycle. Similarly, the RAM also needs to charge and discharge its bitlines for every read operation. In the following sub-sections we discuss our approaches to reduce power in

a CAM/RAM-based issue queue.

3.1 Dynamic Adaptation of the Issue Queue

While fine-grain clock gating is suitable for latch-based issue queues, the shared resources (bitlines, wordlines, taglines, precharge logic, sense amps, *etc.*) of CAM/RAM-based designs make clock gating less effective than for latch-based designs. However, CAM/RAM-based designs are very amenable to *dynamic adaptation* of the issue queue to match application requirements. As described in [2], the size of the issue queue needed to maintain close to peak performance varies from application to application and even among the different phases of a single application. Thus, an issue queue that adapts to these different program phases has the potential to significantly improve power efficiency with little impact on CPI performance.

In this paper, we implement the basic approach proposed in [2]. In this scheme, the issue queue is broken down into multi-entry *chunks*, each of which can be disabled on-the-fly at runtime. A hardware-based monitor measures issue queue activity over a *cycle window* period by counting the number of valid entries in the queue, after which the appropriate control signals disable and enable queue chunks [2].

3.2 Banked Issue Queue

Banking is a common practice for RAM-based structures (*e.g.*, caches) that can both reduce the delay of the RAM and its power dissipation. The essential idea is to break up a single RAM into multiple smaller RAM subarray structures. Although this requires extra predecoding and output multiplexing as well as some duplication of peripheral (precharge, sense amp, and output driver) circuitry, because each subarray is smaller than the original, the overall delay is often decreased. If the division is performed by slicing the bitlines (*i.e.*, by sets), this approach can also reduce power dissipation as only one of the subarrays needs to be activated on each access.

CAM-based structures can also be banked [8], albeit with some potential impact on CPI performance. The low-order n address bits normally used for the comparison are instead used to select one of 2^n CAM subarrays. The remaining bits are compared against the appropriate bits in each CAM subarray entry. Similarly, these n bits are used to pick which subarray a new entry is placed in. Thus, only one of the n subarrays is activated for each CAM access. The CPI degradation comes about when there is a non-uniform usage of the different subarrays, causing some subarrays to become full before others. This inefficient usage of the entries compared to a single CAM structure results in either entries being needlessly replaced or new entries not being able to be inserted even with available space in other subarrays. The result is CPI degradation relative to the single CAM structure.

The issue queue CAM structure presents the additional complication of having two fields (source operand IDs) on which a match operation is performed. To approach the ideal of enabling only one subarray for each access, we propose a novel banked design that exploits the fact that frequently at least one of the two source operands is ready when an instruction is dispatched. Figure 4 shows how frequently only one, both, and neither of the two source operands are ready when instructions are dispatched into the integer queue. The simulation is on six of the SPEC2000 integer programs using the methodology described in Section 4. On average, 13% of the dispatched integer instructions have neither operand ready. The remaining 87% of the instructions have at least one operand available and therefore require at most one match operation for the instruction to wake up.

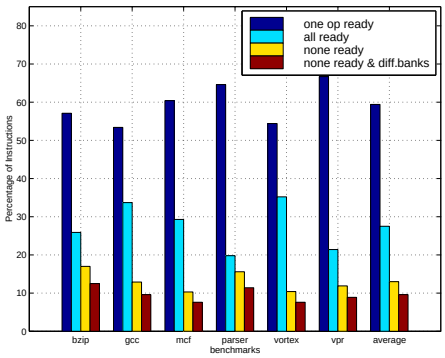

Figure 4: Percentage of instructions with various numbers of operands available on dispatch. Also shown is the percentage of cases in which neither source operand is available and where the source operand IDs are associated with different banks.

A banked issue queue organization that exploit this property is shown in Figure 5. The organization uses four banks, each of which holds two source operand IDs. One is the full six-bit source operand field (assuming 64 physical registers) held in the *instruction info* (RAM) section of the entry while the other consists of only the four low-order register ID bits and is held in the CAM part of the entry (note that 2 of the bits are already used for bank selection). Thus, only the latter is compared against the low-order four destination register ID bits that are broadcast. Thus, our banked issue queue design further reduces power dissipation by eliminating one of the two source operand IDs from the CAM. Note that the match logic is guaranteed to be active for only one cycle. However, the ready logic, selection logic, and the RAM part may be active for more than one cycle. Multiple instructions (say N) may become ready due to result distribution, in which case the ready logic, selection logic, and RAM part may be active for N cycles. The selection logic is global in the sense that instructions may be simultaneously ready in multiple banks.

As shown in the top of Figure 5 for an example *add* instruction, three of the cases of source operands being ready or not on dispatch are easy to handle. The instruction is steered to the bank corresponding to the ID number of the unavailable source operand. In the case where both operands of an instruction are available, the instruction is steered to the bank corresponding to the first operand. An instruction in the selected bank wakes up when there is a match between the lower four bits of the destination ID and those of the source ID corresponding to the unavailable operand. The fourth case, that of neither operand being available on dispatch, is treated as a special case. Here, instructions that have neither source operand available are placed in the *Conflict Queue*. The Conflict Queue is simply a conventional issue queue that performs comparisons with both source operands. Because a small percentage of the instructions have neither source operand available on dispatch, the Conflict Queue need only contain a few entries. The destination IDs of completing instructions are compared with the entries in one of the banks, as well as with those in the Conflict Queue. Because the Conflict Queue is small, its energy dissipation pales in comparison to the savings afforded by banking.

4. METHODOLOGY

The design alternatives are implemented at the circuit level and the power estimations are evaluated by using the IBM AS/X circuit simulation tool with next generation process parameters. All the

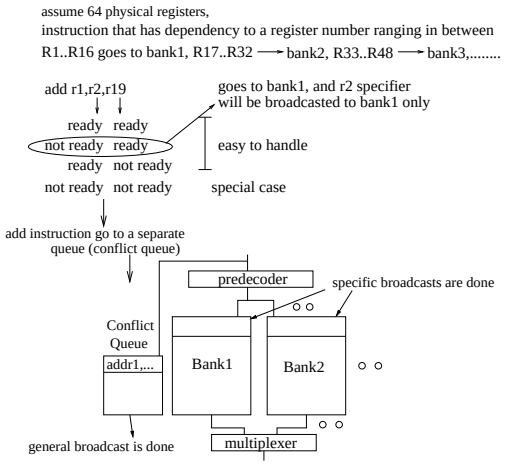

Figure 5: Banked issue queue organization and placement of instructions using the Conflict Queue for the case where neither source operand is available on dispatch.

circuits also have been optimized as much as is reasonable for power and speed. The baseline latch-based issue queue and other circuit designs borrow from existing POWER4 libraries where appropriate.

For the microarchitectural simulations, we used SimpleScalar-3.0 to simulate an aggressive 8-way superscalar out-of-order processor. The simulator has been modified to model separate integer and floating point queues. The baseline also included register renaming and physical registers to properly model banked issue queues. We chose a workload of six of the SPEC2000 integer benchmarks (each of which is run for 400 million instructions). Issue queue event counts are captured during simulation and used with the circuit-level data to estimate power dissipation. We focus on an integer issue queue with 32 entries in this paper, although the techniques are largely applicable to other queue structures (*e.g.*, floating point queue, dispatch queue, reorder buffer). For the simulation parameters, we chose a combined branch predictor of bimodal and 2-level and fetch and decode widths of 16 instructions for our 8-way machine with a reorder buffer size of 128 entries. We used 64KB 2-way L1 and 2MB 4-way L2 caches, four integer ALUs and multipliers and four memory ports.

4.1 Results

For the baseline issue queue[1], each entry needs to be clocked each cycle even when the queue is idle due to the need to recirculate the data through the multiplexer to hold the data in place. In an alternative clock-gated design the main clock as well as the latch clocks are gated by a control signal whenever an entry does not have the Valid bit set and is not being loaded. We first examine the benefits of clock gating the issue queue, which largely depends on what fraction of the entries can be clock gated for our application suite. Figure 6 shows the average number of entries in a 32-entry integer queue that are and are not clock gated as well as the overall power savings achieved. For vortex and gcc, on average over 50% of the queue entries are clock gated, whereas for mcf, parser, and vpr there is not much clock gating opportunity. On average, a 34% power savings is achieved with clock gating the issue queue without any loss of CPI performance.

[1]The baseline described in this paper does not represent the real POWER4 issue queue. Some mechanisms to reduce power, not described in this paper are present in the real POWER4 design.

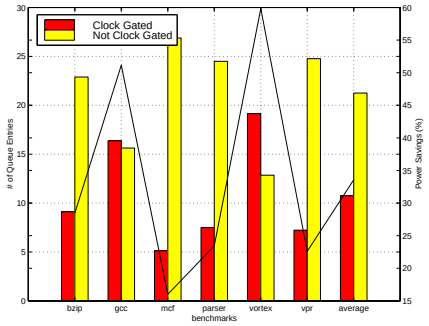

Figure 6: Number of queue entries gated, and power savings relative to baseline for a latch-based issue queue with clock gating.

The tradeoffs between a compacting and non-compacting issue queue are more complex, as a degradation in CPI performance can potentially occur with non-compaction due to the lack of an oldest-first selection scheme. We modified SimpleScalar to model the holes created in a non-compacting issue queue, the filling of these holes with newly dispatched instructions, and a selection mechanism strictly based on location within the queue (rather than the oldest-first mechanism used by default). With such a scheme, older instructions may remain in the queue for a long time period, thereby delaying the completion of important dependence chains. The left-most bar in Figure 7 shows CPI degradation for our six SPEC2000 integer benchmarks. The degradation is significant, around 8% for mcf and parser and 5.5% overall. The right-most bar shows the CPI degradation when the described oldest first selection is implemented by using four bit sequence number (including the sorting bit). On average, the partial oldest first selection scheme reduces the CPI degradation from 5.5% to 2.3%.

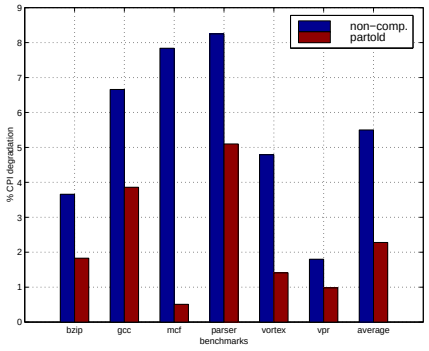

Figure 7: CPI degradation incurred via non-compaction with position-based selection, and non-compaction with partial oldest-first selection.

The power savings of the non-compacting latch-based issue queue relative to the baseline design is shown in Figure 8. The non-compacting queue power includes the power overhead due to the oldest-first selection logic overhead as well as the write arbitration logic overhead that provides the capability of writing to any hole for the newly dispatched instructions. Even with these additional overheads, the elimination of the frequent high-power compacting events has a considerable impact across all benchmarks, achieving a power savings of 25-45% and 36% overall.

This figure also shows the relative power savings of the non-compacting

CAM/RAM-based issue queue, and a non-compacting issue queue implemented with clock gating. Redesigning the issue queue as a CAM/RAM structure achieves a considerable power savings over the non-compacting latch-based design. However, the combination of a non-compacting latch-based design and clock gating achieves slightly better overall savings. Note that the slightly better power savings for mcf, parser, and vpr with the CAM/RAM-based design is due to the lack of opportunity for clock gating with these benchmarks. The choice of one option over the other depends on a number of factors, including the expertise of the design team in terms of clock gating versus CAM/RAM implementation, verification and testing of the CAM/RAM design, and the degree to which the additional clock skew and switching current variations of the clock gated design can be tolerated. In the rest of this section, we explore how the CAM/RAM-based issue queue design can be augmented with dynamic adaptation or banking to further reduce power dissipation.

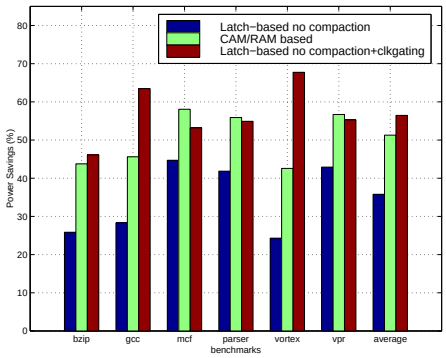

Figure 8: Power savings relative to baseline of non-compacting latch-based, non-compacting CAM/RAM-based, and non-compacting latch-based with clock gating, all with the proposed partial oldest-first selection scheme.

We assume a 32-entry adaptive issue queue that can be configured with 32, 24, 16, or 8 entries during application execution. Figure 9 shows the power savings and performance degradation with the adaptive scheme for different *cycle window* values [2]. Note the negative power savings with mcf using the larger *cycle windows* of 8K and 16K. This occurs because at this coarse level of dynamic adaptation, the 32-entry configuration is always selected which incurs a power penalty due to the overhead of the dynamic adaptation circuitry. The use of smaller *cycle windows* allows the dynamic adaptation algorithm to capture the finer-grain phase change behavior of mcf, resulting in smaller configurations being selected. Over all of these benchmarks, the use of smaller *cycle windows* results in a higher power savings and a lower performance degradation than when larger *cycle windows* are used. For a *cycle window* of 4K, a 34% overall issue queue power savings can be achieved with a 3% CPI degradation as compared to the CAM/RAM-based design.

We explored 2, 4, and 8-way banked issue queues using the Conflict Queue approach described in Section 3.2. The top of Figure 10 shows why banking can be so effective: the relative power of the CAM structure increases quadratically with the number of entries. Banking divides the queue into smaller structures, only one of which is selected each cycle. The bottom part of this figure shows the power savings achieved with different issue queue sizes for 2, 4, and 8-way banked queues with only one bank enabled. There is a clear tradeoff between the reduction in the number of active entries

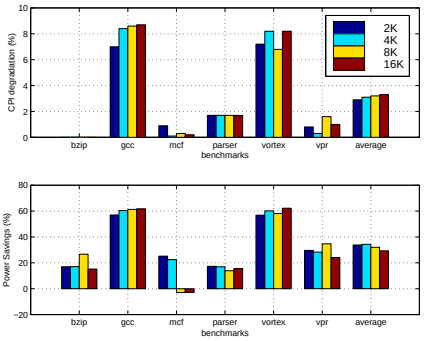

Figure 9: CPI degradation and power savings of the adaptive issue queue relative to the CAM/RAM-based issue queue for different cycle window values.

(and thus bitline length) with higher degrees of banking and the extra peripheral circuit overhead incurred with more banks. For a small queue size of 16 entries, the power savings is greater with two banks due to the relatively large cost of duplicating the peripheral circuitry. With the larger 64 entry queue, the savings in bitline power afforded with 8 banks outweighs the peripheral logic power overhead.

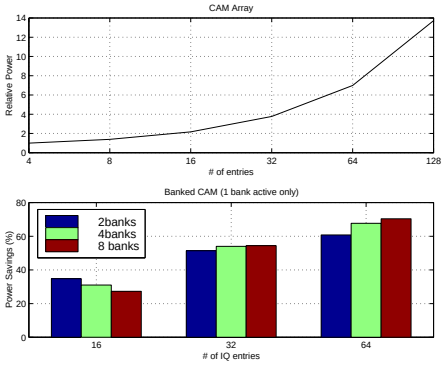

Figure 10: Relative power of the issue queue CAM array as a function of the number of entries (top), and power savings by degree of banking and issue queue size with a single enabled bank (bottom).

As mentioned in Section 3.2, banking can incur a CPI degradation due to underutilization of queue entries resulting from static allocation of dispatched instructions to banks. This can be partially remedied by increasing the number of entries in each bank. The graph at the top of Figure 11 shows the CPI degradation incurred relative to the baseline for a 4-way banked issue queue with a 4-entry Conflict Queue for various numbers of entries per bank. The CPI degradation can be reduced to 2.5% with 10 entries per bank, a slight increase from the 8 entries nominally used. The middle graph shows performance degradation for a 4-way banked queue with 10 entries per bank for different Conflict Queue sizes. A small number of entries (4-5) is sufficient to reduce the CPI degradation to negligible levels. Finally, the bottom graph shows the percentage of time various numbers of banks were active for our 8-way issue machine with a 4-banked issue queue (10 entries per bank, 4 entry Conflict Queue), as well as the power savings achieved for each benchmark. Note that these results account for the power overheads of the extra entries and the Conflict Queue, and we assume that both the

baseline and banked designs have the entire queue disabled with no activity (zero banks active for the banked approach). Overall, a 31% energy savings is achieved with only a 2.5% impact on CPI performance. This compares favorably with the 34% power savings and 3% CPI degradation of the adaptive approach, yet the banked scheme is arguably more straightforward to implement.

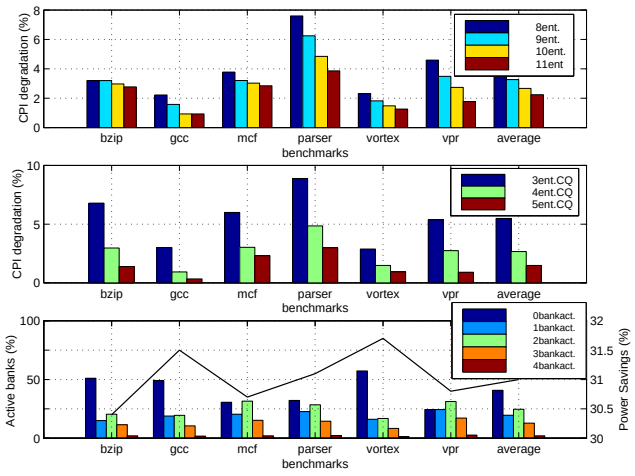

Figure 11: 32-entry four-way banked issue queue results relative to the 32-entry CAM/RAM-based issue queue. CPI degradation with different numbers of entries per bank (top) with a 4 entry Conflict Queue. CPI degradation with different size Conflict Queues (middle) with 10 entries per bank. Percentage of different numbers of active banks and power savings (bottom) with a 4 entry Conflict Queue and 10 entries per bank.

4.2 Comparison of Different Alternatives

Clock gating the issue queue has a significant impact on power dissipation with no CPI degradation. Despite its implementation and verification challenges it is a well-known and established approach and therefore represents the most straightforward, albeit not the most effective, solution to the issue queue power problem. On the other side, making the queue non-compacting affords an even greater power savings, albeit with a CPI performance cost due to the elimination of oldest-first selection. This problem can be largely remedied with the sequence-number and sorting bit scheme proposed in this paper with no delay cost and negligible power impact relative to the power savings with non-compaction. This makes the non-compacting scheme an attractive alternative to the baseline compacting design. The combination of non-compaction and clock gating provides slightly better issue queue power savings than a CAM/RAM-based design. The two alternatives are functionally equivalent, but quite different in terms of a number of implementation and verification cost factors that may favor one over the other.

Once the designer chooses a CAM/RAM-based implementation, an adaptive CAM/RAM-based issue queue delivers an additional 26% power savings beyond non-compaction and clock gating. For designs where every Watt counts, the adaptive approach provides the lowest power dissipation of any of the techniques we examined. However, the cost is a slight performance degradation, in addition to the significant design and verification effort involved. The banked approach with the Conflict Queue represents an attractive alternative to the adaptive design. It's power savings and performance degradation rival that of the adaptive approach, yet its design would be considered more straightforward by most designers. Finally, the banked and adaptive issue queue techniques are orthogonal approaches that can be combined to afford even greater power savings. Due to the size of our issue queue (32 entries) the combination of these techniques would not be profitable. However, a larger 128 entry queue could be divided into four 32 entry banks, each of which would use the adaptive approach described in this paper. Based on our experience and the results in this paper, we expect that this combination would produce much greater power savings than any of the other techniques investigated in this study.

5. CONCLUSIONS

In this paper, we have presented a range of issue queue power optimization techniques that differ in their effectiveness as well as design and verification effort. As part of this study, we propose a sequencing mechanism for non-compacting issue queues that allows for a straightforward implementation of oldest-first selection. We also devised a banked issue queue approach that allows for all but one bank to be disabled with little additional power overhead. Through a detailed quantitative comparison of the techniques, we determine that the combination of a non-compaction scheme and clock gating achieves roughly the same power savings as a CAM/RAM-based issue queue. We also conclude that the adaptive and banked CAM/RAM-based issue queue approaches achieve a significant enough power savings over the latch-based approaches to potentially justify their greater design and verification effort.

6. REFERENCES

[1] P. Bose et al. Early-Stage Definition of LPX: A Low Power Issue-Execute Processor Prototype. PACS02 workshop, HPCA-8, 2002.

[2] A. Buyuktosunoglu et al. An Adaptive Issue Queue for Reduced Power at High-Performance. Springer-Verlag Lecture Notes in Computer Science Vol. 2008: 25-40, November 2000.

[3] G. Kucuk et al. Energy-Efficient Instruction Dispatch Buffer Design for Superscalar Processors. ISLPED-01, 2001.

[4] J. A. Farrell et al. Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor. JSSC, 33(5): 707-712, May 1998.

[5] D. Folegnani et al. Energy-Effective Issue Logic. ISCA-28, 2001.

[6] P. J. Jordan et al. Data Processing System and Method for Using an Unique Identifier to Maintain an Age Relationship Between Executing Instructions. IBM Corporation, U.S. patent 5805849, 1997.

[7] J. Leenstra et al. A 1.8GHz Instruction Window Buffer. ISSCC-01, 2001.

[8] H. Miyatake et al. A Design for High-Speed Low-Power CMOS Fully Parallel Content-Addressable Memory Macros. JSSC, 36(6): 956-968, June 2001.

[9] S. Palacharla et al. Complexity-Effective Superscalar Processors. ISCA-24, 1997.

[10] J. M. Tendler et al. POWER4 System Architecture. http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.pdf, 2001.

[11] K. Wilcox et al. Alpha Processors: A History of Power Issues and a Look to the Future. Cool Chips Tutorial, MICRO-32, 1999.

[12] K. Yeager. The Mips R10000 Superscalar Microprocessor. IEEE Micro, 16(2):28-41, April 1996.