Reducing Issue Queue Power for Multimedia Applications using a Feedback Control Algorithm*

Yu Bai and R. Iris Bahar
Division of Engineering, Brown University, Providence, RI 02912
{Yu_Bai, Iris_Bahar}@Brown.EDU

Abstract

We propose a dynamic power-aware issue queue in a general-purpose microprocessor for multimedia applications. Resources can be adapted at runtime in accordance with the feedback from a formal closed-loop control system such that power dissipation is reduced while still meeting the real time target for multimedia applications. We partition the issue queue into multiple sets (i.e., FIFOs) such that only instructions at the head of each set are able to issue. We then dynamically reconfigure the issue queue by changing the number and/or size of FIFOs to satisfy specific application's needs. The optimal configuration and timing are predicted by the closed-loop control system. Our results show an average power savings of over 90% in the issue queue with a negligible affect on the performance.

1. Introduction

With rapidly developing technology and increasing clock frequency, power dissipation has gained more and more attention. Higher power dissipation could reduce the battery life time for mobile applications, raise the chip costs, and lessen the chip reliability. It could also limit clock frequency and the amount of hardware features that can be included. This study concentrates on reducing power in the issue logic since it is a significant source to the total power dissipation for a modern general-purpose, out-of-order superscalar microprocessor, such as Alpha 21464 [16].

While high-end microprocessor design is mainly driven by performance, power dissipation has become a major design concern. To satisfy needs from the widest set of applications, complex architectural features are included in these general-purpose, high-performance microprocessors. Although the goal of overall high performance is generally met, these features are not optimally utilized mainly because of diversity within and among different applications. During runtime, an application may vary widely in its requirements for different datapath resources, degree of instruction-level parallelism (ILP), branch behavior, and/or memory access behavior. As a result, the datapath resources may not be optimally utilized when the application is running; however, some power will be dissipated by these resources regardless of their utilization.

To address these limitations, we proposed a flexible issue queue, based on [2], that is partitioned into several FIFOs such that only the head of each FIFO is exposed to the arbiter. Moreover, the number and/or size of FIFOs can be adapted during runtime so as to match the running application's dynamic characteristics. In this paper, we focus on real-time multimedia applications since they tend to occupy a large portion of workloads for more and more systems, including mobile or desktop systems that are built using general-purpose microprocessors. Nowadays, it is expected that these systems will be used increasingly for multimedia applications, which usually have real-time deadlines. Specifically, we are focusing on soft real-time multimedia workloads, i.e., missing a small portion of deadlines does not noticeably affect overall system quality. Our solution is to employ a classical feedback control system to adapt the issue queue configuration and save power due to an inherent property of these multimedia applications, i.e., soft real-time deadlines. Our experiments of combining our flexible issue queue scheme and a closed-loop feedback control system show the potentials of reducing issue queue power dissipation while meeting soft real-time deadlines for multimedia applications. In addition, our results demonstrate that this simple feedback system provides a sufficient response speed and is quite stable.

*This research was supported in part by NSF grant number CCR-0311180 and an equipment grant from Sun Microsystems.

2. Related Work

Adjusting available datapath resources to better match program requirements is not new. In terms of when and how resource adaptation is applied, previous research can be divided into two categories: *statically* and *dynamically* adaptive processing.

Static adaptation techniques have been proposed to reduce overall design complexity and/or power by making adaptation decisions before execution and using fixed low-power structures during the whole execution. In [12], the authors introduced a technique called Data-Flow Prescheduling to reduce the complexity of the issue stage by ordering instructions before they enter the issue buffer. Other similar static techniques were proposed in [6]. Palacharla *et al.* analyzed several specific areas of register renaming, instruction window wakeup and arbitration logic, and operand bypassing [13]. They found that the window wakeup and arbitration logic as well as operand bypass are probably the most critical components for future microprocessors and thus will be fundamental for future power-saving research work. Palacharla *et al.* then presented an alternative design with a faster clock and simplified wakeup and arbitration logic which puts chains of dependent instructions into FIFO buffers and issues instructions from multiple buffers in parallel. The drawback of Palacharla's approach, just as with other static techniques that use fixed-sized data structures, is that different applications may not all benefit from only one type of issue queue configuration, thereby limiting its applicability.

A dynamic adaptation organizes adaptive hardware units so that they can rapidly tune their complexity — usually in terms of their size or parallel processing capabilities — to satisfy the current application's needs. In [1], issue width was varied to allow disabling of a cluster of functional units during runtime to save power using feedback from various performance monitors. Other works proposed dynamically reducing the number of active entries in the instruction window according to the processor's needs in order to save power [7, 8, 14]. The shortfall of these approaches is that while dynamically adjusting issue queue size may reduce power in the wakeup and arbitration logic, doing so narrows the scope of instructions available for exposing ILP. This can be potentially harmful to performance when ILP can only be exposed using a large issue queue. Another limitation is that they do not distinguish among valid entries in the issue queue and as such make all of them visible to the wakeup and arbitration logic. This can be very inefficient if instructions remain in the issue queue for many cycles before they are ready to "wake-up" and issue. Our approach addresses these limitations by reconfiguring the is-

sue queue into multiple FIFOs such that instructions dependent on long latency instructions are prevented from being visible to the arbitration logic. We then simultaneously modify the number and size of FIFOs during runtime according to application needs.

Control-theoretic techniques have already been employed to design processor systems. In particular, several methods based on control theory have been proposed to reduce power dissipation, energy consumption, and/or chip temperature. Skadron *et al.* introduced formal feedback control theory into micro-architectural level design as a technique to more adaptively control temperature and minimize performance drop [15]. Real-time deadlines inherent in many multimedia workloads make the use of closed-loop feedback control systems particularly appropriate. The work of [10] focused on how to control dynamic voltage/frequency scaling (DVS) settings for real-time response. Based on a formal control loop, the DVS system was designed to save power without degrading a desired playback rate in a multimedia portable environment. Similarly, [11] proposed a DVS feedback system to reduce power while maintaining the playback rate. Decoder power was reduced by controlling the decoder rate so as to match with the display rate. Our approach is similar to [10] and [11], but we are working on a general-purpose microprocessor and controlling the issue queue configuration such that the system commit rate matches with the commit rate target, which is determined by the multimedia workload.

3. Implementation

3.1 The Dynamically Reconfigurable Issue Queue

Our goal is to dynamically adjust the active size of the issue queue to more closely match a multimedia workload's needs. This effectively reduces the power in the wakeup and arbitration logic of the issue queue. To facilitate this, we follow the approach described in [2, 3]. In particular, we partition the issue queue into several FIFOs such that only the instructions at the head of each FIFO are visible to the arbitration logic. Therefore, each FIFO issues in-order though overall instruction issue among different FIFOs is out-of-order. In Figure 1, we show an example of the *Hybrid Scheme* as proposed in [3]. In the first stage, it starts to reconfigure the issue queue by modifying both the number and size of FIFOs while keeping the entire issue queue active. When the FIFO size reaches some pre-defined threshold, in the second stage, some FIFOs may be completely disabled if determined that they are not needed to retain performance.

Given that only a fraction of the entries will be visible to the arbitration logic, it is important that instructions be placed in the FIFOs such that most (if not all) of the ready instructions appear at the head of a FIFO. Otherwise, performance is more likely to suffer. We chose to implement a dependency-based scheme similar to the one presented in [13]. As an instruction is decoded and dispatched to the issue queue, the scheme attempts to place it in the same FIFO as one or both of its source dependencies.

3.2 The Closed-Loop Feedback System

The architecture of the whole pipeline is shown in Figure 2. To realize a control-theoretic system, we add three main components: *Commit Buffer*, *Compare Logic*, and *Controller*. The main goal of our closed-loop feedback system is to control the system's actual commit rate to be as close as possible to the *Commit Rate Target* that reflects the requirement of the specific multimedia application. To facilitate this goal, a *Commit Buffer* is added, which is shown on the right side of Figure 2. The *Commit Buffer* is a small buffer (e.g., 20 entries) used to hold retired instructions in

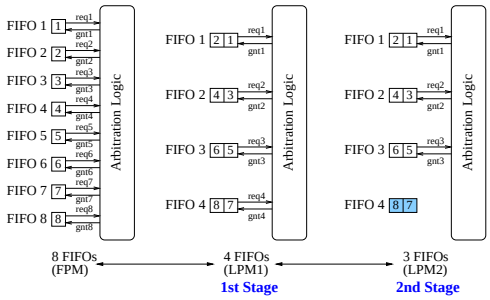

Figure 1. A dynamically reconfigurable issue queue.

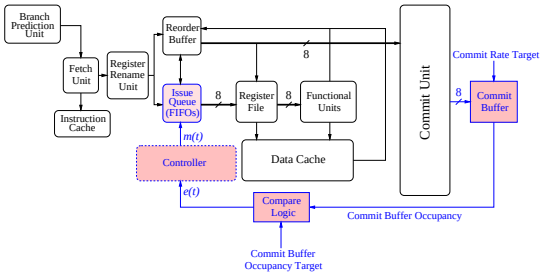

Figure 2. Pipeline organization.

program order.[1] Instructions leave the buffer at the rate defined in *Commit Rate Target*. When there are fewer instructions than *Commit Rate Target* in the *Commit Buffer*, a target miss occurs. Our ultimate goal is to adjust the number of active issue queue resources dynamically, such that power dissipation is minimized and target performance is obtained at all times. However, as mentioned in Section 1, we are only focusing on soft real-time multimedia workloads; therefore missing a target commit rate may be reasonable as long as this occurs a small fraction of the time.

To control the system's commit rate, we measure the occupancy of the *Commit Buffer*, compare it with *Commit Buffer Occupancy Target* (i.e., set-point), and send the error (i.e., $e(t)$ in the figure) to a classical controller to adapt the configuration of the issue queue such that available resources match the workload's dynamic requirements. *Commit Buffer Occupancy Target* is given by an interval: $[C_l, C_h]$. When comparing *Commit Buffer*'s actual occupancy with *Commit Buffer Occupancy Target* in *Compare Logic*, three cases are possible: (1) if *Commit Buffer*'s actual occupancy is smaller than C_l, issue queue resources are insufficient to meet the target performance and the error is set to $e(t) = C_l - actual\ occupancy$; (2) if *Commit Buffer*'s actual occupancy is larger than C_h, issue queue resources are excessive and the error is to $e(t) = C_h - actual\ occupancy$; (3) if *Commit Buffer*'s actual occupancy is in the range $[C_l, C_h]$, the issue queue configuration is appropriate and thus the error is set to be zero. We experimented with different intervals for *Commit Buffer Occupancy Target* (i.e., [2, 6], [3, 8], and [4, 10]). We found that results are not sensitive to the specific interval used and thus [3, 8] is assumed for the following experiments.

The classical *proportional, integral, derivative* (PID) controller corrects the error simultaneously by considering present, past, and future conditions according to the equation:

$$m(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \qquad (1)$$

[1] It is not strictly required to move instructions to the *Commit Buffer* once they are ready to retire, but doing so allows us to decouple the feedback system from the rest of the pipeline; otherwise, we may need to stall the machine because the reorder buffer might hold instructions that are ready to retire for a longer time.

where $m(t)$ provides the control for resizing the issue queue in the near future. The *proportional action* handles the present, by multiplying the error by a proportional constant K_P such that the error is reduced. The *integral action* improves stability by integrating (or summing) the error over a time period, and then multiplying this value by a constant K_I to handle the past. To handle the future, the *derivative action* calculates the rate of change for the error with respect to time, and then multiplies this value by a constant K_D to reduce the change rate of the error. It turns out that for our purposes we can obtain reasonable results by considering only proportional and integral actions. In theory, a strict system analysis is needed to identify the best action constants. Unfortunately, we cannot provide an accurate system model due to its complexity. However, the small constants tend to make the system stable, but with slower response. Again, our experiments show that small constants, which are simply selected without formal system analysis, work well enough to meet our goals.

Implementing this closed-loop feedback system on chip requires a relatively small amount of hardware, including a small buffer, a few comparators, registers, small-bit adders and shifters. Therefore, we assume that the feedback system itself has a negligible impact on power dissipation.

4. Power Estimations

The FIFO-based issue queue design saves power by turning off under-utilized issue queue components in the instruction wakeup and arbitration loop. To make our power analysis straightforward, we only consider the power-saving estimations in the issue queue. We assume that any significant savings in the issue queue will translate to the reasonable savings in the processor since the issue queue is a significant contributor to the total power dissipation on the chip [4, 16]. To estimate the power savings in the issue queue when operating in a low-power mode, we used the same approach as was presented in [3] and extrapolated from Alpha 21264 and 21464 power estimations [4, 16]. As with our design, the 21464 was designed with a unified, non-collapsible, out-of-order issue queue, capable of 8-wide issue.

The wakeup and arbitration loop consists of the generation of data ready, instruction request, and instruction grant signals. Due to its complicated hardware implementation and control logic, this loop is a critical path in the processor design, and is therefore responsible for a significant portion of the issue queue power dissipation [4]. Depending on different needs and implementation strategies, the power distribution within the wakeup and arbitration loop may vary [3]. For example, if a more power-efficient wakeup design is implemented, half of the loop power may be attributed to the wakeup logic and the other half to the arbitration logic [4]. We call this distribution "*DISTR1*". If a faster, but more power-hungry, wakeup design is employed, the power distribution could be 70% in the wakeup logic and the remaining 30% in the arbitration logic. This is referred to as "*DISTR2*".

In the first stage of our scheme, we reconfigure the issue queue by dynamically modifying the number and size of each FIFO while keeping all entries in the issue queue active at all times. The power dissipation of the arbitration logic is reduced since only instructions at the head of a FIFO will be visible to the arbiter. However, the issue queue entry still needs to update the dependency information and readiness of instruction operands. By this reasoning, we assume that in the first stage, power dissipation in the wakeup logic remains the same regardless of the FIFO configuration. Power dissipation in the arbitration logic is saved due to the reduced activity on the request and grant lines by selectively inhibiting them from precharging. We only need to precharge the request and grant lines for instructions that are at the head of a FIFO. So the power dissipation in the arbitration logic is directly proportional to the number of active FIFOs. If the total number of FIFO heads in the issue queue is cut in half, this should reduce the switching and therefore the power on the request and grant lines and associated logic by approximately half.

In the second stage, we deactivate FIFOs if they are found to be under-utilized. We do not need to update any information associated with disabled entries. Thus, we can save power in the wakeup logic according to the fraction of FIFOs disabled. Power dissipation in the arbitration logic is also reduced due to the reduced activity on the corresponding request and grant lines. Moreover, interconnect wire capacitance can be reduced in the remaining active request and grant lines by isolating the portion of the wires routed through the disabled entries.

5. Experimental Methodology

Our simulator is derived from the SIMPLESCALAR tool suite [5] executing PISA (Portable ISA) binaries. We added several modifications to SIMPLESCALAR to better model our reconfigurable processor. In particular, we split the RUU into a reorder buffer (ROB) and issue queue (IQ), thus allowing us to more accurately model current and next generation processors and enabling us to implement the issue queue with multiple FIFOs.

Our simulations model a pipeline that allows for up to 8 instructions to be issued, executed, and committed each cycle. In addition, the base case assumes a unified 128-entry, out-of-order issue queue plus a reorder buffer and load/store queue with 512 and 256 entries respectively to avoid having it become a performance bottleneck. First level and second level caches are 64KB, 2-way associative with 3 cycle latency, and 256KB, 8-way associative with 12 cycle latency, respectively. Access to main memory takes 150 cycles. Our simulations are executed on a subset of the multimedia benchmarks from MediaBench [9]. All benchmarks were compiled using a re-targeted version of the GNU *gcc* compiler with full optimization. The benchmarks are executed for 500 million committed instructions, or until they complete, whichever comes first.

6. Experimental Results

We chose the *Commit Rate Target* value by experimenting with different targets (i.e., 0.3, 0.5, 1, and 2) using a conventional issue queue, and observing the percentage of time that the target is missed. Not surprisingly, we found that the more aggressive the target, the more frequently we missed the target. Our target is set to be as high as possible such that we miss the target only a small percentage of the time (e.g., 0.5%). Table 1 shows the commit rate targets that we chose and percentage of time that the corresponding target is missed. This way we can find the target upper bound; a lower target would lead to more power savings, but by definition has lower performance. Thus power savings shown in this paper is the lower bound that our proposed scheme can reach.

Now that we have a *Commit Rate Target* that can be obtained using a conventional, monolithic issue queue, we apply our scheme

Table 1. *Commit Rate Target* **used in this work.**

Benchmarks	*Commit Rate Target*	Miss Percentage
rawcaudio/rawdaudio	0.5/0.5	0.1%/0.1%
cjpeg/djpeg	0.5/1	0.3%/0.1%
mesa	1	0.4%
mpeg2dec/mpeg2enc	2/0.5	0.5%/0.0%
pegwit	0.5	0.2%
epic/unepic	0.5/0.33	0.2%/0.3%

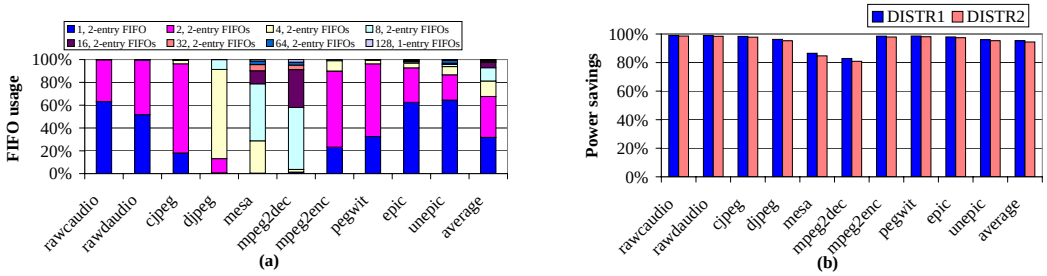

Figure 3. FIFO usage (a) and power savings (b) in the wakeup and arbitration loop for a 128-entry issue queue initially.

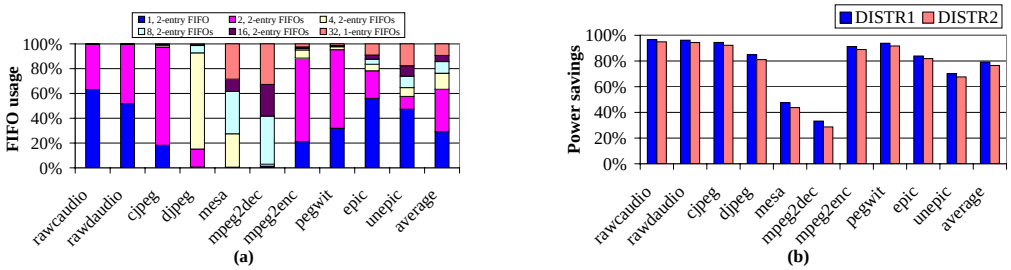

Figure 4. FIFO usage (a) and power savings (b) in the wakeup and arbitration loop for a 32-entry issue queue initially.

to save power while nearly always maintaining this target. Multimedia workloads are not the only applications running on general-purpose microprocessors. In order to satisfy needs from the broadest set of applications, complex architectural features and more datapath resources are included in these microprocessors. Therefore, we start with an aggressive issue queue with 128 entries. Figure 3(a) shows the percentage of time that the processor spends in each mode. Note that we start with a traditional issue queue with 128, 1-entry FIFOs. In the first stage, we reconfigure the issue queue by cutting the number of FIFOs in half, i.e., 64, 2-entry FIFOs. After that, the system enters the second stage since the maximal FIFO size is defined as 2. Overall, we can spend an average of 32% and 36% of the time in the lowest two power modes, which have 2, 2-entry FIFOs and 1, 2-entry FIFO. Using results from Figure 3(a), and power estimations made in Section 4, we can estimate the total power savings in the wakeup and arbitration loop for both distributions, which is shown in Figure 3(b). All results are compared to a monolithic, 128-entry issue queue. As can be seen, we can save at least 80% of power dissipation in the wakeup and arbitration loop and overall we can save 95.3% (for *DISTR1*) or 94.4% (for *DISTR2*) of the total power in the loop, with only 1.7% of time that the targets are missed, on average.

For contrast, we also ran experiments starting with a 32-entry issue queue. Similarly, Figure 4 shows the FIFO usage and power savings in the loop. Power results are compared with a conventional, 32-entry issue queue. As expected, we stay in the low power modes for a smaller percentage of time and save less power than when starting with a 128-entry issue queue. However, our experimental results still show an average power reduction of 79.2% (for *DISTR1*) or 76.5% (for *DISTR2*) in the wakeup and arbitration loop within the issue queue with an average 1.4% of the time that the targets are missed. We also found that around the same amount of power in the loop is dissipated to complete each application when starting with either a large issue queue or a small one, which indicates that our proposed system is effective in finding an appropriate configuration for different applications regardless of the base configuration.

7. Conclusion

Power dissipation is wasted for applications that do not need the most aggressive features, such as multimedia applications. We provide a flexible solution by applying a simple and stable closed-loop feedback system with a dynamically reconfigurable mixed in-order/out-of-order issue queue design. Our simulations show great potential savings in power for the issue queue while still meeting the performance demands inherent in multimedia applications.

8. REFERENCES

[1] R. I. Bahar and S. Manne. Power and Energy Reduction Via Pipeline Balancing. In *ISCA*, July 2001.

[2] Y. Bai and R. I. Bahar. A Dynamically Reconfigurable Mixed In-Order/Out-of-Order Issue Queue for Power-Aware Microprocessors. In *ISVLSI*, February 2003.

[3] Y. Bai and R. I. Bahar. A Low Power In-Order/Out-of-Order Issue Queue. *ACM Transactions on Architecture and Code Optimization*, 1(2), June 2004.

[4] L. Biro. Intel Corporation, November 2003. Private communication.

[5] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical report, University of Wisconsin, Madison, June 1999.

[6] R. Canal and A. González. Reducing the Complexity of the Issue Logic. In *ICS*, June 2001.

[7] D. Folegnani and A. González. Energy-Effective Issue Logic. In *ISCA*, July 2001.

[8] C. J. Hughes and S. V. Adve. Spreading Slack for Optimal Energy-Performance Tradeoffs for Multimedia Applications. In *ISCA*, June 2004.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *MICRO*, December 1997.

[10] Z. Lu, J. Hein, M. Humphrey, M. R. Stan, J. Lach, and K. Skadron. Control-Theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads. In *CASES*, October 2002.

[11] Z. Lu, J. Lach, M. R. Stan, and K. Skadron. Reducing Multimedia Decode Power using Feedback Control. In *ICCD*, October 2003.

[12] P. Michaud and A. Seznec. Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors. In *HPCA*, January 2001.

[13] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *ISCA*, June 1997.

[14] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources. In *MICRO*, December 2001.

[15] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *HPCA02*, February 2002.

[16] K. Wilcox and S. Manne. Alpha Processors: A History of Power Issues and a Look to the Future. In *Cool-Chips Tutorial*, November 1999.