A Centralized Cache Miss Driven Technique to Improve Processor Power Dissipation

Houman Homayoun[‡], Mohammad Makhzan[†], Jean-Luc Gaudiot[†], Alex Veidenbaum[‡]
[†]*Department of Electrical and Computer Engineering, UC Irvine*
[‡]*Department of Computer Science, UC Irvine*
{hhomayou,alexv}@ics.uci.edu {mmakhzan,gaudiot}@uci.edu

***Abstract*--Leakage and Dynamic power are a major challenge in microprocessor design. Many circuit techniques along with micro-architectural innovations have been proposed to reduce power in individual processor units But it is not clear that these techniques can be combined. A centralized approach which can reduce power in more than one unit at a time with minimal the hardware overhead is needed.**

This paper proposes such a centralized approach that attempts to simultaneously reduce power in processor units with highest dissipation: the reorder buffer, the instruction queue, and the integer and the floating-point register files. It is based on an observation that utilization for the aforementioned units varies significantly, during a period when an L2 cache miss or multiple L1 cache misses are pending as compared to a period when none of these are present. Therefore we propose to dynamically adjust the size and thus power dissipation of these resources during such periods. Circuit level modifications required for such resource adaptation are presented. Simulation results for SPEC2K benchmarks show a substantial reduction in both leakage and dynamic power: the total dynamic power is reduced by as much as 30, 31, 31 and 48% for the reorder buffer, the integer register file, the floating-point register file and the instruction queue, respectively. The reduction in leakage is up to 33% for reorder buffer and 37% for integer and floating-point register files. The total energy-delay product is reduced, on average, by 15, 26, 20 and 17% for the reorder buffer, the integer register file, the floating-point register file and the instruction queue respectively. This comes at the cost of a performance impact which is as low as 0.9% for integer and 2.2% for floating-point benchmarks. The required hardware modification is shown to be minimal.

I. INTRODUCTION

Both dynamic and leakage power dissipation are a major challenge in designing new processors, in particularly for deep sub-micron technology (65nm and below).

For many individual processor units, several power reduction techniques have been proposed in the literature. Attempts have been made to either design new power-efficient units or to make current architectures more power-aware. However, the prior efforts have resulted in approaches which require considerable re-design and verification efforts. Further, it is not clear that these techniques can be combined, and, if they can be combined, that the power and energy-delay savings would still be considerable and whether the cumulative of the performance degradation and complexity they individually introduce would still be negligible. The challenge is thus to find a *centralized* and simple algorithm which can reduce power for more than one unit (and ultimately the entire chip) and comes with the least amount of re-design and verification efforts, the lowest possible design risk (which would come with any new design) and the least hardware overhead. Current industry trends towards deploying multi simple-cores processor on a single chip (*multicore* chips) emphasize the demand for such simple centralized solutions for power conservations rather than complex mechanisms.

In recent years, several efforts have sought such centralized algorithm through two major approaches: either dynamically adapting the data-path resources for power conservation [18, 20, 21, 29] or dynamically adapting the voltage and frequency level at a fine granularity or at the entire chip [2, 4, 5]. These techniques have several drawbacks: first, many of these techniques are unable to meet the performance requirements of high-end computing; for instance a 8.5% and a 20% performance loss were reported in [18, 2] respectively. Second, many of these techniques introduce significant complexity and overhead: for instance cycle-by-cycle monitoring of program IPC, floating-point IPC, resource utilizations, commit rate or a combination of these [18, 20, 21, 29]), which make them difficult to implement in practice. In addition, circuit assist to deploy such architectural approach are less studied. This is particularly important for approaches using resource size adaptation since their effectiveness is influenced by the power/delay transition overhead associated when resizing resources. Such transition overhead is largely determined by a circuit implementation.

The research presented in this paper falls into the first category and investigates dynamic recourses adaptation for power reduction. Unlike previous approaches which require continuous cycle-by-cycle monitoring of resource occupancy to predict future resource needs, our approach is deterministic rather than predictive. It relies on the fact that processor resource utilization *deterministically* vary significantly when cache miss(es) occur, especially the L2 cache miss(es). There is thus no need for expensive cycle-by-cycle monitoring of processor resource occupancy. In addition, our technique requires minimal hardware modification.

The approach proposed in this paper aims to reduce power in the reorder buffer (*ROB*), the instruction queue (IQ), the integer register file (*IRF*), and the floating-point register file (*FRF*) at the same time but with minimal hardware changes. Novel circuit techniques are presented to accomplish the proposed architectural strategy. It is based on the observation that processor performance drops significantly after an L2 cache miss. Similarly, a considerable performance reduction occurs during any period in which multiple L1 misses are

978-1-4244-1985-2/08/$25.00 ©2008 IEEE

pending. Indeed, during cache miss periods, the processor needs significantly lower issue/wakeup width, but a larger ROB and register files. Thus we propose to dynamically adjust the issue/ wake up width, and the size of the reorder buffer and of the *IRF* and *FRF*.

This paper makes four major contributions:

- It demonstrates a substantial, deterministic variation in processor resource utilization when one or more L2 cache misses or at least three L1 cache misses are pending (the *cache miss period*) as compared to when none of these conditions are present,
- It presents a control algorithm to dynamically adjust the size of these units during cache miss periods for power conservation,
- It shows the minimal required hardware modifications to dynamically adjust the size of these units,
- It presents Spice simulation results for the instruction queue (using CACTI 4.0 for ROB and register files) and shows a substantial reduction in both leakage and dynamic power at negligible or no performance cost.

The rest of the paper is organized as follows: related work and background are described in Section II. Section III presents the motivation for proposed architectural techniques. Section IV describes our proposed architectural technique. We describe the issue/wakeup mechanism, *ROB* and register files functionality and their major sources of complexity and power consumption. We also discuss the circuit modification required to implement our architectural algorithm. Evaluation methodology is described in Sec. V, experimental results in Sec. VI and conclusions in Sec. VII.

II. RELATED WORK AND BACKGROUND

There is a significant body of work on the design of the Instruction Queue, of the ROB, and of the register file. Indeed, several techniques have been proposed to reduce their power expenditure. Yet, as we shall show, much improvement can still be achieved by our technique.

A. Instruction Queue

The Instruction Queue is a CAM-like structure which holds instructions until they can be issued. Four possible actions are associated with it:

1) Set an entry for a new dispatched instruction,
2) Read an entry to issue an instruction to a functional unit,
3) Wakeup instructions waiting in the IQ once a result is ready, and
4) Select instructions for issue when the instructions available exceed the processor issue limit (to which we refer as issue width or IW for short).

The main complexity of the Instruction Queue stems from the associative search during the wakeup process. All above tasks are energy demanding and make the Instruction Queue one of the major energy consumers in the processor as shown in [12, 22, 30].

B. ROB and Register File

The *ROB* and the register file are multi-ported SRAM structures with many functions:

1) Setting entries for up to IW instructions in each cycle,
2) Releasing up to IW entries during commit stage in a cycle,
3) And flushing entries during the branch recovery.

It is easy to recognize that, combined, these functions have high power dissipation (estimated to dissipate as much as 27% and 16%, respectively, of the total chip power [17]).

C. Power Reduction

There has been a significant body of work on reducing power in a single data-path component such as Instruction Queue, the ROB, and the register file. Most recently, Canal and Gonzalez [22] have proposed a scheme which schedules instructions based on their expected issue time. Homayoun *et al.* introduced the idea of lazy instructions and propose to selectively wakeup them once predicted [12]. Unlike the above algorithms which require substantial modifications or even complete re-design, our proposed architecture requires only minimal modification of a conventional instruction queue logic; using gated-Vdd transistor to power gate the match lines appropriately, and yet it is highly effective in reducing instruction queue power.

In [16], it has been proposed to partition the *ROB* into independent units, each with a separate pre-charge, sense amp, input and output drivers and the ability to activate or deactivate each based on a continuous monitoring of the processor IPC. Our approach adaptively resizes the *ROB,* but does not require continuous monitoring of IPC (and thus does not require additional hardware and associated power overhead). Second, our proposed circuit requires minimal hardware modifications; an AND logic and a couple of pass transistors and a gated VSS transistor, unlike [16] which requires a separate sense amp, peripheral drivers and pre-charge units which require considerable modifications. Third, the performance loss associated with our proposed technique is less than that in [16], while it is incurred when the power reduction algorithm is applied to other units as well; IQ and *IRF* and *FRF*.

Past work on the design of the register file mostly either attempt to limit the number of ports or limit its size [7, 8, 9, 11, 17, 19]. In [17], it has been proposed to bypass the register information that will be used from the fetch stage to the decode stage, hence putting unused registers into the low power mode in the early pipeline stage. To avoid substantial performance penalty, the registers had to be put back to high power mode one cycle before being accessed. There are also proposals for reducing the number of ports at the cost of additional arbitration hardware. These techniques require substantial modifications to the pipeline. Borch *et al.* have discussed problems associated with reducing the number of register file

ports [6]. Further, banked register files, caching registers, and using 2-level register files have been investigated to reduce the number of registers and accordingly the required power [6, 11, 24]. Many of these techniques are based on speculation.

Relaxing physical register file release before the commit stage and before all its consumers have read it has been studied in [7].The drawback of these techniques is in the complexity that speculation adds and more specifically the complexity they introduce on handling the coherency in register caches and banking conflicts.

There have also been some recent projects dealing with centralized techniques which tackle power reduction in multiple data-path components, either by dynamically adapting the data-path resources for power conservation [18, 20, 21, 29] or dynamically adapting the voltage and frequency level at a fine granularity or at the entire chip level [2,4, 5].

Ponemarev *et al.* [18, 29] proposed to monitor processor resource occupancies (infrequently) on which they base an estimate for future requirements. In fact this does not always result in correct estimation and thus it can incur performance penalty. Indeed, mis-predictions mostly occurs when an L2/multiple L1 cache miss/es occurs during which processor resource occupancy grows significantly (as we will show in this paper) and is not necessarily correlated to its past behavior. This results in a significant performance degradation for benchmarks with high L2 cache miss rate such as *vortex* and *applu* (8% and 14% performance loss reported in [18, 29] respectively). Thus it is important to take L1 and L2 cache miss rate into consideration for resource adaptation.

Bahar and Manne [21] studied resource adaptation for a multi-clustered Compaq 21264 processor for which the dispatch rate can varies between 2, 4, and 6 to allow the unused clustered to be shut off. Such variations are triggered when the overall and floating-point IPC pass a threshold and require a significant overhead of cycle by cycle monitoring dispatch unit, IPC, etc. The power reduction of reorder buffer has not been explored in this work.

Li et al. have proposed to apply voltage scaling during L2 cache-miss service time [5]. In fact applying voltage scaling for such small period is not practical. For instance as reported in [10] applying voltage scaling to the Intel Xscale requires 20 microseconds which translates to thousands of processor cycles. This is far more than a few hundred cycles of L2 cache miss service time.

III. MOTIVATION

A load instruction missing in the cache (DL1 or L2) prevents any dependent instruction from being issued. Dependent instructions thus fill up the reorder buffer, the instruction queue, the register files, and/or the load and store queues (LQ/SQ) until the miss returns. We now briefly study the *ROB*, the instruction queue and the register file status during such period for baseline architecture. At each cycle, up to IW (in our case 4) instructions are dispatched to the *ROB* and up to IW physical registers are being allocated out of the pool of free registers. To allocate new instructions, the processor releases up to IW committed-instruction physical registers and their *ROB* entries. This is being done in program order to handle precise interrupts. When a cache miss occurs, the load miss instruction stays on top of the *ROB* and does not allow any subsequent instruction to be committed. As a result, the allocated *ROB* and register files entries for subsequent instructions cannot be released. Thus, while the processor dispatched up to IW instructions at each cycle, it cannot release the *ROB* and the register file entries for the subsequent instructions until the miss returns. This will gradually increase the occupancy of the *ROB* and of the register files and consequently reduce the processor issue rate. The same scenario occurs for LQ/SQ and IQ: the subsequent dependent instruction to the load cache miss cannot be issued due to the data dependency. Such instructions reside in the IQ until the miss returns. Accordingly, the IQ occupancy increases but due to data dependencies, very few out of these can be issued.

Given the long cache miss service time (20 cycles for DL1 and 300 cycles for L2 in our architecture), the above scenario can happen quite frequently. In the event of a L2 miss, due to the long service time, either *ROB*, LQ/SQ or instruction queue fills up with subsequent instructions and the processor ends up stalled (issue rate = 0) until the miss is serviced. We refer to this as scenario I.

In the event of a DL1 miss, the service time is much smaller than it would be for L2 and it is less likely that any of LQ/SQ, *ROB* and IQ (all referred as queues) fills up before the cache miss is serviced. Note that when a DL1 cache miss occurs, its dependent instructions cannot be issued and that all the subsequent instructions cannot be committed as discussed above. This reduces the issue rate and increases the occupancy of the aforementioned queues. In the presence of many pending DL1 cache misses, the impact on the issue rate could be large. Also, the occupancy of queues increases significantly. We refer to the case with multiple DL1 misses pending as scenario II. We refer to the period during which one or more L2 miss/misses and/or multiple DL1 misses are pending as the "cache miss period," and to the rest of program execution time as the "normal period."

It should be noted that the two scenarios discussed above would occur when the missed load is part of a correct prediction path, otherwise after the correct path has been identified, the missed load instruction will be flushed and will release *ROB*/IQ/ LQ/SQ entries so that program execution can continue (return to the normal period).

Figures 1 and Table I, show statistics for the above discussed structure during a cache miss period. Fig. 1, presents the issue rate reduction for scenario I compared to when there is no pending L2 miss and issue rate reduction for scenario II compared to the period where there are less than 3 pending DL1 misses. The issue rate decreases significantly in both cases. Across all benchmarks, the issue rate drops by more than 80% for scenario I. For the case of 3 DL1 misses, the reduction across benchmarks varies significantly; from 60% for facerec to around -1% for gcc and swim. The average is a 22%

reduction for integer benchmarks and 32.6% reduction for floating-point benchmarks.

The reduction in issue rate during the cache miss period, as shown above, results in a gradual increase in the occupancy of ROB (reported in Table I). For integer benchmarks, the ROB occupancy grows substantially; 98.2% for scenario I and 61.4%

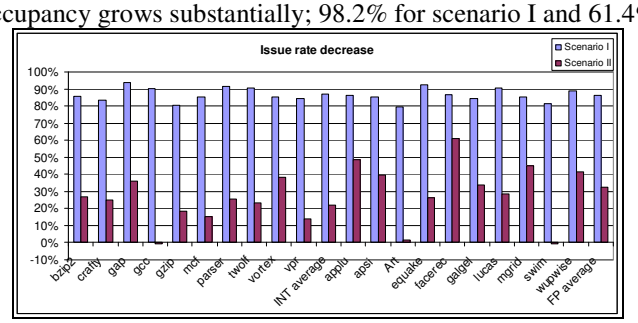


Figure 1. Issue rate decrease for scenario I and II.

for scenario II. This growth is less significant for floating-point benchmarks: 30.5% for scenario I and 25% for scenario II. As expected, for most benchmarks, the ROB occupancy is smaller for scenario II compared to I. This is consistent with the results in Figure 1: the reduction in issue rate results in the ROB occupancy increase.

Table 1. Relative *ROB* occupancy increase during cache miss period.

***ROB* occupancy increase**	**Scenario I**	**Scenario II**		**Scenario I**	**Scenario II**
bzip2	165.0	88.6	applu	13.8	-4.9
crafty	179.6	63.6	apsi	46.6	18.2
gap	16.6	61.7	Art	31.7	56.9
gcc	97.7	43.9	equake	49.8	38.1
gzip	152.9	41.0	facerec	87.9	14.1
mcf	42.2	40.6	galgel	30.9	34.4
parser	31.3	102.3	lucas	-0.7	54.0
twolf	81.8	58.8	mgrid	8.8	5.6
vortex	118.7	57.8	swim	-4.3	11.4
vpr	96.6	55.7	wupwise	40.2	24.4
INT average	***98.2***	***61.4***	***FP average***	***30.5***	***25.2***

Our simulation results (not shown due to space limitation) show that the *IRF* occupancy always grows for both scenarios when experimenting with integer benchmarks. There is also a similar case for *FRF* when running floating-point benchmarks and only during scenario II. For the remaining cases, there is significant variation across benchmarks (98% decreases to 580% increases). This is particularly the case for *IRF* with floating-point benchmarks and for *FRF* with integer benchmarks. Based on the results shown, in the next section we propose a simple algorithm to reduce the power dissipation in the *ROB*, the Register Files and the issue/wakeup units.

IV. PROPOSED CENTRALIZED APPROACH

The approach proposed in this paper aim to reduce the dynamic and static power dissipation of the *ROB*, the Register Files, and the Instruction Queue, by adaptive resizing. Reducing the size of any of these units will require different hardware modifications (which come at extra power/area cost) which are decided by the resizing scheme. In order to minimize the hardware cost, we propose a simple resizing scheme and go from normal to half size and back. While this is not the most optimal resizing scheme for individual units it is the simplest one in terms of hardware modifications.

We propose to reduce the issue and the wakeup width of the processor only during L2 miss service times (scenario I). The results in Figure 1 show that the issue width for scenario II is also reduced but not as significantly as it would have been in scenario I. We are thus trying not to impact the IPC by reducing the issue width for scenario II. Our experimental results confirm this conjecture and show that the performance degradation is not negligible.

Based on a significant increase in *ROB* occupancy during cache miss periods, we propose to increase its size during such periods (during both scenarios I and II). During normal periods, we keep the *ROB* size at half its possible size.

The dynamic adaptation of register file requires more caution since the FRF and IRF behave differently for integer and floating-point benchmarks. Based on what discussed for IRF we propose to increase its size during both scenarios I and II and when running integer benchmarks. We propose to apply a similar technique for FRF when running floating point benchmark. In addition, as explained the IRF occupancy when running floating point benchmarks is relatively small. Thus we can reduce its size when running floating point benchmarks. A similar case is true for FRF when running integer benchmarks.

After the cache miss period ends (start of normal period) the ROB and register file occupancies decrease. We therefore propose to reduce the size of ROB and register file by haf after a cache miss period ends and once the augmented half part is become empty.

A. Reducing the Effective Width of Issue and Wakeup

We propose reducing the size of the wakeup and issue width from 4 to 2. In Figure 2 shows the circuit level implementation for one row of the instruction queue. At each cycle, the match lines are pre-charged high which allows the individual bits associated with an instruction tag to be compared with the results broadcasted on the taglines. Upon a mismatch, the corresponding matchline is discharged. Otherwise, the match line stays at Vdd, which indicates a tag match. At each cycle, since we may have up to 4 instructions broadcasted on the taglines, we need to have four sets of one-bit comparators for each one-bit cell, as shown in Figure 2. All four matchlines must be ORed together to detect a match on any of the broadcasted tags. The result of the OR sets the ready bit of instruction source operand showing that it is ready. As shown in [14, 22], our results confirm that matchline discharge is the major energy consumption activity responsible for more than 58% of the energy consumption in the instruction queue. As the matchline must go across the entire width of the instruction queue, it has a large wire capacitance. Adding the one-bit comparators diffusion capacitance makes the equivalent capacitance of matchline large. Pre-charging and discharging this large capacitor is responsible for the majority of power consumption in the instruction queue.

Previous studies have shown that a broadcasted tag has on average one dependent instruction in the instruction queue [27]. In other words, most of the time, all the instruction queue matchlines are discharged except one. For our configuration, among 32x4 matchlines in the instruction queue on average in each cycle, only one is not discharged. Discharging the other matchlines will cause significant power dissipation in the instruction queue.

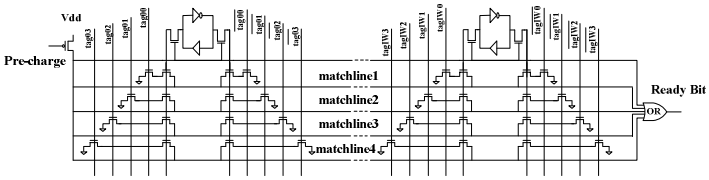


Figure 2. Circuit implementation of instruction queue.

Results in Figure 1 showed the average issue rate (and of course, the wakeup rate) to be far less than one for scenario I. In other words, out of four taglines, on average only one carries the broadcasted data from the functional units to all entries in the instruction queue. This means that pre-charging the other matchlines is not useful. We can thus avoid pre-charging such matchlines by using a gated-Vdd transistor as shown in Figure 3(b). Therefore, the question is how to determine which set of matchlines associated with specific tagline should be disabled rather than pre-charged. The matchlines are being pre-charged when the clock is low and are conditionally discharged immediately after the results have been broadcasted on the taglines (when the clock is high). Pre-charging the matchline has to be done before the tags are broadcasted on the taglines. In order to meet this deadline, we need to know the matchlines associated with taglines which do not carry tags. This allows us to disable them. In Figure 3(b) we show the circuit modification needed to reduce the wakeup width from 4 to 2 and hence the disabling/pre-charging and discharging of half of the matchlines. By multiplexing the data bus to taglines 1 and 3 we can safely turn off other matchlines associated with the rest of the taglines. It should be noted that the multiplexing is done only during those periods for which the average issue/wakeup width is small; multiplexing the data bus over taglines when the average issue rate is more than two can significantly degrade the performance..

The worth case scenario in our design is the case in which more than half of taglines are broadcasting tags during scenario I where only half of matchlines are active. To respond to such an event, we buffer (refer to as auxiliary broadcast buffer) half of the tags and broadcast them whenever a tagline associated with an active matchline becomes available. Considering the very low average issue/wakeup rate during scenario I, such worse case scenario happens very rarely. While it is extremely rare, it is also possible that the auxiliary broadcast buffer becomes full. Our experimental results show that a 4-entry broadcast buffer is sufficient to minimize such occurrence. In the very rare case that the broadcast buffer fills up, the functional unit execution is stalled until the broadcast buffer become available.

Reducing the wakeup/issue width during normal program execution (when no cache miss is pending) requires a large number of auxiliary broadcast buffers and comes at the cost of a large power overhead which could adversely impact the performance. This is another reason why we only apply dynamic issue and wakeup width adaptation during scenario I.

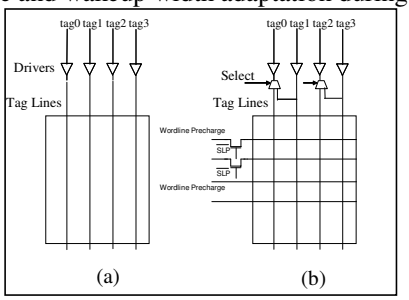


Figure 3. (a) Baseline issue/wake up logic (b) modified issue/wake up logic.

B. Dynamically Resizing ROB and Register Files

Figure 4 shows the circuit level implementation of an SRAM. To read or write an entry each cycle all the bit lines must be pre-charged high (fired). For write operations, the high voltage on the bit lines induces a logic 1 into a cell for which the word line is fired. To read the content of an entry, one of bit line or $\overline{bitline}$ will be conditionally discharged. The sense amplifier detects such a difference and will drive it to the output buffer. Bit lines thus must run across the entire *ROB* or register file height. As we may have multiple accesses to the same cell in a cycle, the read bit line and the write bit line thus need to be separated [28]. Hence, if we are to read N entries at each cycle and write to W entries in the same cycle, we must have (N + W) * 2 bit lines. In our design of register files, we have (8 + 4) * 2 * 64 (data width is 64 bits) bit lines for register files and (4+4) * 2 * 100 bit lines for *ROB*. Pre-charging and discharging this large a number of bit lines is a major source of power dissipation in these two structures [16, 28].

Figure 5 shows the breakdown of dynamic energy (for read operations) and leakage power of the register file components. The bit lines are the major consumers of power. It should be noted that most of the leakage of bit lines is due to the leakage currents of memory cells, which flow through the two *off* pass transistor to the bit lines. Accordingly, by eliminating the leakage in memory cells, we can eliminate the bit line leakage. As shown, the *ROB* and register file utilization is relatively low. Hence, one approach to reduce power dissipation in these two units would be to turn off the unused entries and their associated wordline drivers using circuit techniques such as gated-Vdd or gated Vss and eliminating the leakage power dissipation virtually completely (sleep mode). The transition from sleep to active mode adds a one-cycle delay to the *ROB* or register file access which has significant performance impact. The algorithms proposed in the previous section reduce the performance impact of frequently activating and deactivating the entries. Resizing the *ROB* could be achieved by partitioning it into several independent units with separate sense amps, pre-chargers, input output drivers as explained in [16]. [16], proposed to partition the *ROB* into 8 units. This requires 8 times more sense amps pre-charge lines and input and output drivers compared to the non-partitioned structure. The cost in

terms of power and area is not negligible. To avoid adding to the complexity of *ROB* and register files, we use the divided bit line technique [26] proposed for SRAMs to reduce the bit line capacitance and hence its dynamic power..

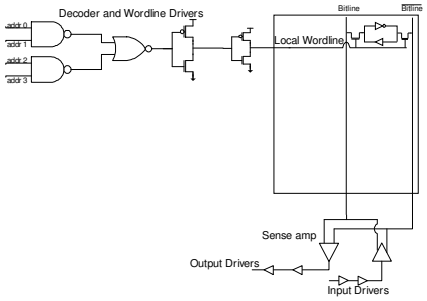
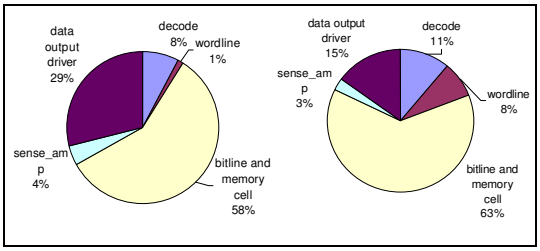


Figure 4. *ROB* and Register File SRAM Circuit



(a) (b)

Figure 5. (a) Dynamic energy (b) leakage power of the register file.

As shown in Figure 6,m two or more SRAM cells are combined together to divide the bit line into several sub-bit lines. In the non-divided bit line structure the bit line capacitance is *N * diffusion capacitance of pass transistors + wire capacitance* (usually 10% of total diffusion capacitance) where N is the total number of rows (in our case 128 for *ROB* and 128 for register files). In the divided bit line scheme the equivalent bit line capacitance is reduced to *M * diffusion capacitance + wire capacitance*, where M is the number of bit line segments (sub-bit lines). As bit line dynamic power dissipation is proportional to cv^2, reducing the effective capacitance would linearly reduces the bit line dynamic power. It should be noted that the overhead of this technique is adding a set of pass transistors per sub-bit line (shown in Figure 6 as segment control switch). As a side effect, the large number of segments increases the area and power overhead.

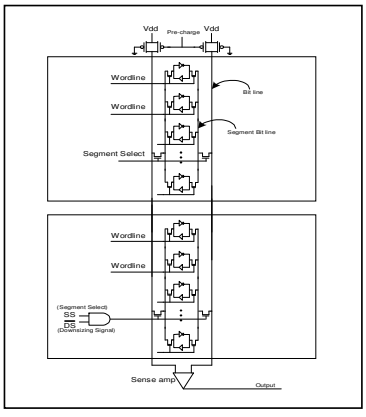


Figure 6. Divided bit line circuit.

Since this technique is incorporated in CACTI [15], we used the toolset to find the best number of bit line segments for area and power optimizations. We found 8 bit line segments for register files and the ROB results in minimal area and power overhead. To downsize the ROB, the select signal of the lower partition is being AND together with the downsize signal. Doing that, no write and read can be done to/from the partition and the entire partition can be turned off safely. To turn off the entire partition we use the gated Vdd technique [25] to suppress the voltage in all memory cells of the partition and eliminating its leakage almost completely. We also use a similar technique to eliminate leakage in the wordline driver of the disabled partition. Beginning of a cache miss period triggers upsizing the unit by de-asserting the downsize signal and turning on the disabled partition. The overhead of downsizing and upsizing is 1 cycle (gated-Vdd overhead). The end of cache miss period triggers downsizing the ROB. Note that the downsize signal is asserted only when the segment is empty.

The benefits of such resizing is in reducing both dynamic and leakage power. Leakage is suppressed by turning off the entire segment of memory cells and wordline driver. Dynamic power is reduced due to a smaller equivalent capacitance on the bit lines. The same hardware modification is applied to register files.

The same approach applied to ROB is being applied to IRF and FRF when running integer and floating point benchmarks respectively. In addition, the downsize signal is kept asserted always for IRF when running floating point benchmarks and for FRF when running integer benchmarks It should be noted that Once the cache miss period ends and the augmented half (lower partition) becomes empty, the size of ROB and register file is reduced back to half of their size. This requires detecting when the lower partition becomes empty after the end of cache miss period. This can be accomplished by using an additional *bit* in each row (entry) of the lower partition. This bit is set when an entry in the lower partition is used (register write) and reset when the entry is released (during commit).

V. EXPERIMENTAL METHODOLOGY

To evaluate the proposed approach, we estimate the leakage and dynamic power reduction, the total energy-delay reduction, and the IPC change. Table II describes the processor architecture, the clock frequency is 2GHz. SPEC2K benchmarks were compiled with the O4 flag using the Compaq compiler for the Alpha 21264 processor and executed with reference data sets.

Table II. Processor organization

L1 I-cache	128KB, 64 byte/line, 2 cycles	**Instruction queue**	64 entry (32 INT and 32 FP)
L1 D-cache	128KB, 64 byte/line, 2 cycles, 2 R/W ports	**Register file**	128 integer and 128 floating-point
L2 cache	4MB, 8 way, 64 byte/line, 20 cycles	**Load/store queue**	32 entry load and 32 entry store
issue	4 way out of order	**Arithmetic unit**	4 integer, 4 fp units
Branch predictor	64KB entry g-share,4K-entry BTB	**Complex unit**	2 INT, 2 FP multiply/divide units
Reorder buffer	128 entry	**Pipeline**	15 cycles (some stages are multi-cycles)

The architecture was simulated using an extensively modified version of SimpleScalar 4.0 [23]. The benchmarks were fast–forwarded for 2 billion instructions, then fully simulated for 2 billion instructions. A modified version of Cacti4 [15] was used for estimating power in the *ROB* and the Register files in 65nm technology. The power in the Instruction Queue was evaluated using Spice and the TSMC 65nm technology with Vdd at 1.08 volts.

VI. RESULTS

Power savings and performance changes associated with our approach are shown in Figures 7 and 8. The approach is used for the *ROB*, register files, and issue/wakeup width simultaneously. Figure 7 shows the fraction of execution time when the *ROB*, *IRF* and *FRF* and issue/wakeup width were reduced to half their size for each benchmark.

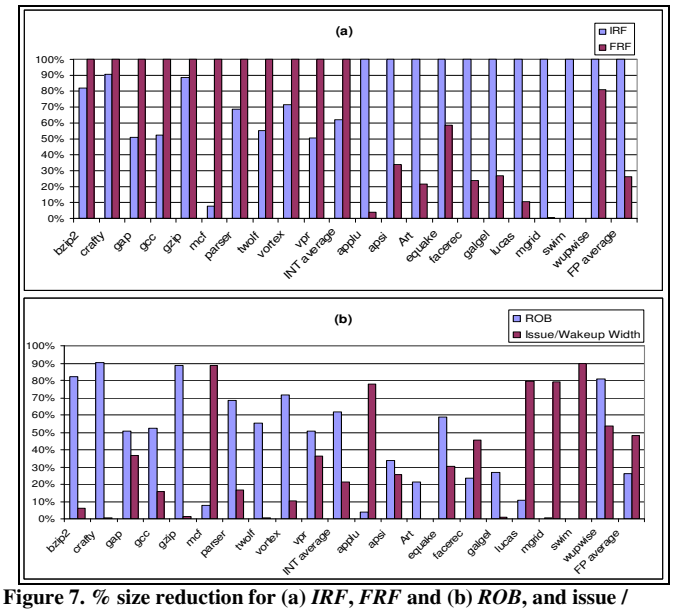


Figure 7. % size reduction for (a) *IRF*, *FRF* and (b) *ROB*, and issue / wakeup width.

IPC reduction is shown in Figure 8. Leakage and dynamic power reduction for individual units is shown in Figure 9. On average, the performance loss is 0.9% for integer benchmarks and 2.2% for floating-point benchmarks. The issue/wakeup width is reduced from 4 to 2 for 21% of integer benchmarks' life time (maximum is 88% for *mcf*). It is higher for floating-point benchmarks: 48% on average (maximum 90% for *swim*). This difference means that integer benchmarks consume more issue bandwidth compared to floating-point benchmarks.

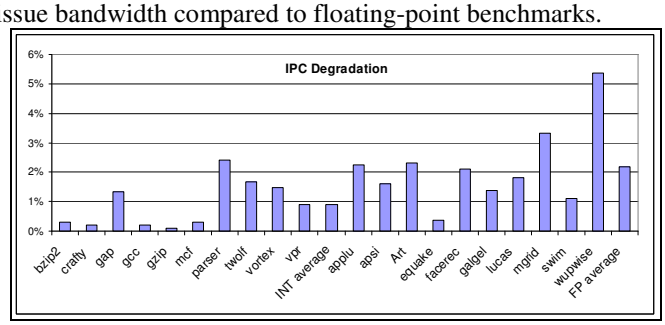


Figure 8. IPC degradation due to *resource resizing*

The same Figure show that *ROB* and Integer register file are kept in the low power mode for 51% of integer benchmarks life time, while *ROB* and *FRF* are kept in the low power mode for 26% of the floating-point benchmarks lifetime. As explained earlier, our proposed algorithm keeps *FRF* and *IRF* in the low power mode for the entire lifetime of integer benchmarks and floating-point benchmarks respectively (bars showing 100% in Figure 7).

Figure 10 shows the energy-delay product reduction. In most benchmarks, our technique reduces the total energy-delay product by up to 48% (in IQ for *mcf*). Some of the benchmarks show a slight increase in their energy-delay. For ROB and FRF, this is the case for *lucas* and *mgrid* with 3% and 1% increase respectively. For the instruction queue, the energy-delay product increases for *twolf*, *apsi* and *facrec* with 1.5%, 2.3% and 0.8% respectively. For *IRF* the energy-delay decreases across all benchmarks. On average the total energy-delay product decreases by 15, 26, 20 and 17% for *ROB*, *IRF*, *FRF* and instruction queue respectively.

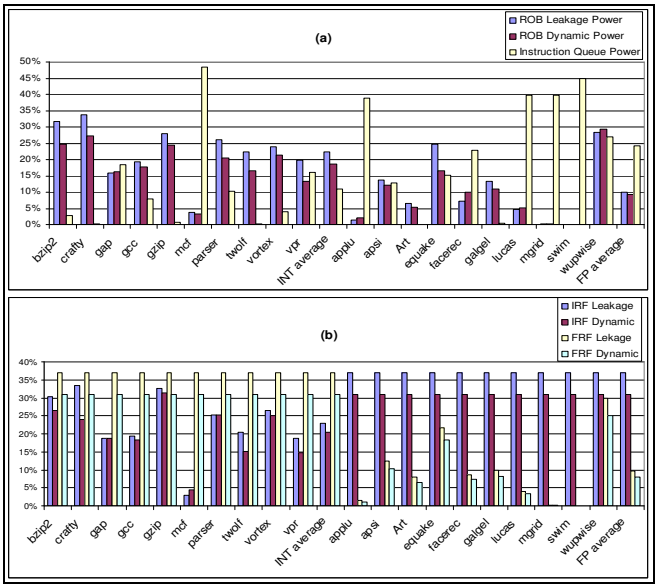


Figure 9. Dynamic and Leakage Power reduction in (a) *ROB* and Instruction Queue and (b) *IRF* and *FRF*.

In floating-point benchmarks the instruction queue benefits considerably from our technique: power reduction reaches 45% (24% on the average). The average savings are 11% for integer benchmarks. For reorder buffer, our technique affects more integer benchmarks; 19% dynamic power reduction and 23% leakage power savings. As for floating-point benchmarks, the leakage and dynamic power savings reach 10% and 9% respectively. The average dynamic and leakage power savings (floating-point and integer benchmarks) for *IRF* is 26% and 30% respectively (20% and 24% for *FRF)*.

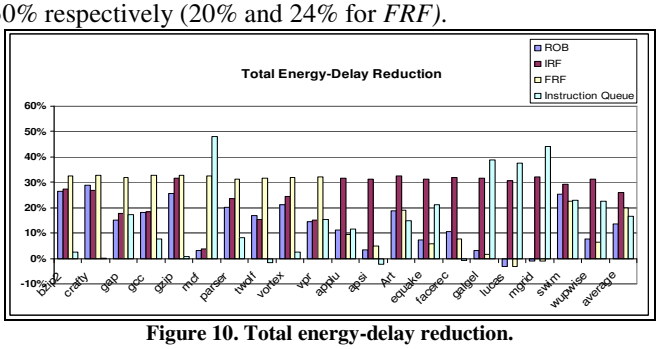


Figure 10. Total energy-delay reduction.

Applying the new algorithm, the total energy-delay product is reduced, on average, by 15%, 26%, 20% and 17% for the reorder buffer, the integer register file, the floating-point register file and the instruction queue, respectively, for SPEC2K benchmarks. This comes at the cost of a 0.9% and 2.2% performance loss for integer and floating-point benchmark, respectively.

Our future work is to investigate ways to utilize this algorithm for other processor structures, such as BTB, load/store queue, L1 and L2 caches.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under Grant CCF-0541403. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US National Science Foundation.

REFERENCES

[1] R. Gonzalez, A. Cristal, A. Veidenbaum, and M. Valero, "A Content Aware Register File Organization", Proc. 31st International Symposium on Computer Architecture (ISCA04).

[2] K. Choi, R. Soma, and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff Based on the Ratio of Off-Chip Access to On-Chip Computation Times". *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 1, January 2005*.

[3] Marco A. Ramirez, Adrian Cristal, Alexander V. Veidenbaum, Luis Villa, Mateo Valero. A New Pointer-based Instruction Queue Design and Its Power-Performance Evaluation. Proc. of the IEEE Int'l Conference on Computer Design (ICCD-2005).

[4] C. Hsu and W. Feng "Effective Dynamic Voltage Scaling through CPU-Boundedness Detection". *4th IEEE/ACM Workshop on Power-Aware Computer Systems, December 2004*.

[5] H. Li, C.-Y. Cher, T. Vijaykumar, and K. Roy. "VSV: L2-miss-driven variable supply-voltage scaling for low power." *International Symposium on Microarchitecture , December 2003*.

[6] E. Borch, E. Tune, S. Manne, and J. Emer, "Loose loops sink chips," *In Eighth International Symposium on High Performance Computer Architecture*, pages 299–310, Feb. 2002.

[7] D. Balkan, J. Sharkey, D. Ponomarev and A. Aggarwal, "Address- Value decoupling for early register deallocation," *Proc. 35th Int'l Conf. on Parallel Processing (ICPP-06).*

[8] I. Park, M. D. Powell, and T. N. Vijaykumar, "Reducing register ports for higher speed and lower energy," *In MICRO-35*, Istanbul, Turkey, November 2002.

[9] H. Homayoun, S. Pasricha, M. Makhzan, A. Veidenbaum, "Improving Performance and Reducing Energy-Delay with Adaptive Resource Resizing for Out-of-Order Embedded Processors". ACM SIGPLAN/SIGBED 2008 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2008). Tucson, Arizona.

[10] M. Fleischmann. "Crusoe Power Management: Cutting x86 Operating Power Through LongRun." *Embedded Processor Forum, June 2000*.

[11] J. H. Tseng and Krste Asanović, "Banked multiported register files for high-frequency superscalar microprocessors," *In 30th intl. symposium on Computer architecture*, June 2003.

[12] H. Homayoun and A. Baniasadi, "Using lazy instruction prediction to reduce processor wakeup power dissipation." *The 2nd workshop on unique chips and systems, in conjunction with the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2006).*

[13] E. Tune, R. Kumar D.M. Tullsen, and B. Calder, "Balanced multithreading: Increasing throughput via a low cost multithreading hierarchy," *In Proceedings of The 37th Annual International Symposium on Microarchitecture (MICRO-37 2004),* 4-8 December 2004, Portland, OR, USA, pages 183–194. IEEE Computer Society, 2004.

[14] A. Buyuktosunoglu, D. H. Albonesi, P. Bose, P. W. Cook and S. E. Schuster, "Tradeoffs in power-efficient issue queue design," *In Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, Aug. 2002.

[15] "Cacti4," http://quid.hpl.hp.com:9081/cacti/.

[16] D. Ponomarev, G. Kucuk and K. Ghose, "Energy–Efficient design of the reorder buffer," *PATMOS'02* Seville, Spain, September 2002.

[17] J.L. Ayala, M. Lopez-Vallejo, A. Veidenbaum and C.A. Lopez, "Energy aware register file implementation through instruction predecode," *Proceedings IEEE International Conference On Application-specific Systems, Architectures, and Processors (ASIP03).* June 2003

[18] D. Ponomarev G. Kucuk, K. Ghose, "Dynamic Allocation of Datapath Resources for Low Power." *in Proc. Workshop on Complexity-Effective Design (WCED'01), held in conjunction with ISCA 2001.*

[19] H. Homayoun, S. Pasricha, M. Makhzan, A. Veidenbaum, "Dynamic Register File Resizing and Frequency Scaling to Improve Embedded Processor Performance and Energy-Delay Efficiency", 45th Design Automation Conference, Anaheim California, 2008.

[20] D. Albonesi, et al, "Dynamically Tuning Processor Resources with Adaptive Processing", *IEEE Computer, Special Issue on Power-Aware Computing, Dec. 2003.*

[21] I. Bahar and Srilatha Manne, "Power and Energy Reduction Via Pipeline Balancing", *Proc. International Symposium on Computer Architecture (ISCA), 2001*.

[22] R. Canal and A. Gonzalez, "Reducing the complexity of the issue logic," *In Proceedings of 2001 International Conferences on Supercomputing,* June 2001.

[23] "SimpleScalar4 tutorial", SimpleScalar LLC. http://www.simplescalar.com/tutorial.html.

[24] R. Balasubramonian, S. Dwarkadas and D. H. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processors," *MICRO 2001*.

[25] M.D. Powell, S. Yang, B. Falsafi, K. Roy and T.N. Vijaykumar, "Gated Vdd: A circuit technique to reduce leakage in deep-submicron cache memories," *In Proc. IEEE ISLPED*, 2000.

[26] A. Karandikar and K. K. Parhi, "Low power SRAM design using hierarchical divided bit-line approach," Prec. *Of International Conference on Computer Design*, 1998.

[27] M. Huang, J. Renau and J. Torrellas, "Energy-efficient hybrid wakeup logic," *Proceedings of the 2002 International Symposium on Low Power Electronics and Design* Aug. 2002.

[28] V. V. Zyuban and P. M. Kogge, "The energy complexity of register files." *1998 International Symposium on Low Power Electronics and Design*.

[29] D. Ponomarev, G Kucuk, K. Ghose "Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources", *Proc. 34th IEEE/ACM International Symposium on Microarchitecture (MICRO-34), 2001*.

[30] J. S. Hu, N. Vijaykrishnan and M. J. Irwin, "Exploring wakeup-free instruction scheduling," *In Proceedings of the 10th International Conference on High-Performance Computer Architecture (HPCA-10 2004),* 14-18 February 2004, Madrid, Spain