Reducing Power in All Major CAM and SRAM-Based Processor Units via Centralized, Dynamic Resource Size Management

Houman Homayoun, *Member, IEEE*, Avesta Sasan, *Student Member, IEEE*, Jean-Luc Gaudiot, *Fellow, IEEE*, and Alex Veidenbaum, *Member, IEEE*

***Abstract*—Power minimization has become a primary concern in microprocessor design. In recent years, many circuit and micro-architectural innovations have been proposed to reduce power in many individual processor units. However, many of these prior efforts have concentrated on the approaches which require considerable redesign and verification efforts. Also it has not been investigated whether these techniques can be combined. Therefore a challenge is to find a centralized and simple algorithm which can address power issues for more than one unit, and ultimately the entire chip and comes with the least amount of redesign and verification efforts, the lowest possible design risk and the least hardware overhead. This paper proposes such a centralized approach that attempts to simultaneously reduce power in processor units with highest dissipation: reorder buffer, instruction queue, load/store queue, and register files. It is based on an observation that utilization for the aforementioned units varies significantly, during cache miss period. 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 show a substantial power reduction at the cost of a negligible performance impact and a small hardware overhead.**

***Index Terms*—Cache miss driven, CAM unit, centralized low power technique, dynamic resource resizing, SRAM unit.**

I. Introduction

LEAKAGE and dynamic power has grown significantly and is a major challenge in microprocessor design. In particular, in deep sub-micrometer technology (65 nm and below) high power dissipation become a major concern. 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 redesign 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 performance degradation and complexity they individually introduce would still be negligible. The challenge is thus to find a *centralized* and simple mechanism which can reduce power for more than one unit (and ultimately the entire chip) and comes with the least amount of redesign and verification efforts, the lowest possible design risk (which comes with any new design) and the least hardware overhead. Current industry trends towards deploying multisimple cores on a single chip (*multicore* chips) emphasize the demand for such simple centralized solutions for power conservation rather than complex mechanisms. An example is the next generation Intel processor, Nehalem, a multicore processor with up to eight cores. In Intel Nehalem a new block is introduced referred to as power control unit or in brief PCU to implement a centralized power optimization mechanism [50].

In recent years, several efforts have sought such a 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 for 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 [2] and [18], respectively. Second, many of these techniques introduce significant complexity and overhead: for instance cycle-by-cycle monitoring of program instruction per cycle (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 approaches are studied less. This is particularly important for approaches using resource size adaptation since their effectiveness is influenced by the power/delay transition overhead associated with resizing resources. Such transition overhead is largely determined by the 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

Manuscript received December 25, 2009; revised April 28, 2010; accepted July 11, 2010. This work was supported in part by the U.S. National Science Foundation under Grant CCF-0541403 and Grant CCF-0811882.

H. Homayoun is with the Department of Computer Science, UCI, Irvine, CA 92617 USA.

A. Sasan is with the Department of Electrical Engineering and Computer Science, UCI, Irvine, CA 92617 USA.

J.-L. Gaudiot is with the Department of Electrical and Computer Engineering, University of California at Irvine, Irvine, CA 92697 USA.

A. Veidenbaum is with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697-3425 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2010.2064185

than predictive. It relies on the fact that processor resource utilization varies *deterministically* significantly when cache misses occur, especially the L2 cache misses. 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 load queue (LQ), and store queue (SQ), 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 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, LQ, SQ, and of the IRF and FRF. Such resource adaptation does not come free and impacts performance in some benchmarks.

This paper makes the following five major contributions.

- It demonstrates a substantial, deterministic increase 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 centralized control algorithm based on cache miss information to dynamically adjust the size of these units during cache miss periods for power conservation.
- It proposes to reduce the issue/wake up width, the number of entries of load/store queue, reorder buffer and register file dynamically during cache miss period.
- It shows the minimal required hardware modifications to dynamically adjust the size of these units.
- It presents SPICE simulation results for the instruction queue and the load/store 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 this 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, LQ, SQ, 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 Section V, experimental results in Section VI and the conclusion in Section VII.

II. Related Work and Background

There is a significant body of work on the design of the instruction queue, load/store 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+RAM-like structure which holds instructions until they can be issued. The following 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;
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], [33]–[37], and [51].

B. Reorder Buffer

The *reorder buffer* is a multiported SRAM structures with the following 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) 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. Register File

The pipeline of an out-of-order processor is capable of fetching, decoding, renaming several instructions per processor clock cycle. The processor can also execute and later commit up to as many instructions in each cycle as the issue width. This type of out-of-order multiple-issue processor accesses the register file very frequently. Up to $2\times$ IW reads and IW writes can be issued to the register file per clock cycle. The physical register file is typically designed as a SRAM structure with as many write and twice as many read ports as the maximum number of instructions the processor can issue in each cycle. Thus the register file is one of the most active components in a processor.

Such frequent accesses make the register file one of the hottest units in a processor due to the large power that is dissipated in a small size SRAM structure [45]–[47]. This makes the register file prone to "heat stroke" [49]. Heat stroke occurs when the temperature of any of processor units exceeds beyond a critical limit. To reduce the chances of a heat stroke it is crucial to reduce the power of the register file.

D. Load/Store Queue

The load/store queue is a CAM+RAM structure that supports simultaneous associative searches for maintaining memory consistency and memory dependency. Conventionally store queue is searched associatively to perform in-flight load store forwarding. The following three possible actions are associated with load queue and store queue and store queue:

1) set an entry and keep the order of newly dispatched load/store instructions;
2) associative search of the store queue for a possible load-store forwarding and find the most recent store value;

3) associative searches of the load queue to find and squash premature loads (when there is a store-load order violation).

The main complexity of the load queue and store queue stems from the associative search. This has been reported in several recent works [38]–[42].

E. Power Reduction

There has been a significant body of work on reducing power in a single processor 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 redesign, our proposed architecture requires only minimal modification of a conventional instruction queue logic. It uses gated-Vdd transistor to power gate the match lines appropriately, and yet it is highly effective in reducing instruction queue power.

In [51], Aggarwal *et al.* proposed to decrease wakeup width such that it is smaller than the machine width. This is based on the observation that the full processor width is rarely being used for instruction wakeup process. They present a detailed circuit-level modification required to implement reduced wakeup width design. While our circuit level solution to implement reduced wakeup width is similar to their approach, unlike their approach where wakeup width is statically reduced, we dynamically adjust the wakeup width based on benchmark behavior and thus have more flexibility.

In [16], it has been proposed to partition the ROB into independent units, each with a separate precharge, 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. [16] required a separate sense amp, peripheral drivers, and precharge units which means considerable modifications. Last but not least, the performance loss associated with our proposed technique is less than that in [16].

Past work on the design of the register file mostly either attempt to limit the number of ports or limit its size [7]–[9], [11], [17], [19]. In [17], it has been proposed to bypass the register data 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 two-level register files have been investigated to reduce the number of registers and accordingly the required power [6], [11], [24], [38], [39]. In a recent paper [31], Alastruey *et al.* proposed adding an auxiliary off core register file to support speculative register renaming. Relaxing physical register file release before the commit stage and before all its consumers have read it has been studied in [7] and [32]. Many of these techniques are based on speculation. 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 a performance penalty. Indeed, mispredictions mostly occur when an L2/multiple L1 cache miss/es occur 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] and [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 multiclustered Compaq 21264 processor for which the dispatch rate can vary 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 of 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 μs which translates to thousands of processor cycles. This is far more than a few hundred cycles of L2 cache miss service time.

There are also several works that have attempted to reduce the power dissipation of the load/store queue by reducing the number of associative searches. Sethumadhavan *et al.* [40] proposed using Single-bit hash tables via Bloom filters to reduce the number of searches. Chrysos *et al.* [41] proposed using a store-set predictor to reduce the search requirements. Park *et al.* used similar methodology and implemented a load buffer for out-of-order loads that reduces the number of load queue searches and increasing the size of the load queue and store queue by using segmentation. Sethumadhavan *et al.* [42] proposed to allocate entries in the LQ and SQ at issue instead of decode stage. They reduced the size of the load/store queue, but their technique requires new algorithm to handle overflows. Several works have developed techniques to reduce the pressure on

the Store Queue based on speculative memory bypassing SMB [43]. However, a design that predicts all store-load forwardings [43], [44] requires a non-associative store queue.

III. Motivation

A load instruction missing in a 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. Let us briefly consider the ROB, the instruction queue and the register file behavior 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 which is similar to Intel Core 2 Duo processor), 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 $(\text{issue rate} = 0)$ 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 for L2 and it is less likely that any of RF, LQ/SQ, ROB, and IQ (all referred to as queues) fill 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.[1] 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 where at least three DL1 misses are pending as scenario II (one or two pending DL1 miss/es does not necessarily increase RF, LQ/SQ, ROB, and IQ occupancy as well as issue rate noticeably). 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."

[1]Out-of-order processors use speculative techniques such as load-hit speculation to improve performance [52], [53]. While in this environment a dependent instruction to a DL1 miss can be issued the overall observation made in this section regarding reduction in processor issue rate remains valid.

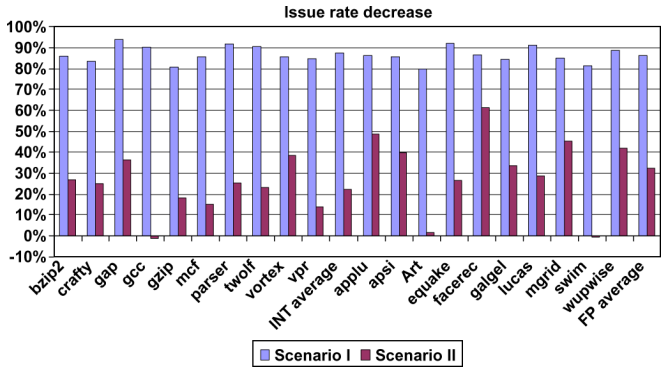


Fig. 1. Percent issue rate decrease for scenario I and II relative to normal period.

It should be noted that the two scenarios discussed above would occur when the missed load is part of a correctly predicted 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).

We studied processor units occupancies during cache miss period for a 4-wide processor with 15 stages of pipeline, 128-entry reorder buffer and register file, 64-entry instruction queue, and a separate 32-entry load and 32-entry store queue (more information is presented in Section V). Fig. 1 and Tables I–III, show statistics for the structures discussed above during a cache miss period for SPEC2K benchmarks. We measured ROB, RF, IQ, LQ, and SQ occupancy as well as issue rate during scenario I (at least one pending L2 miss), scenario II (at least three pending DL1 misses), non-scenario I (no pending L2 miss) and non-scenario II (zero, one or two pending DL1 misses). 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 a 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% for scenario II. This growth is less significant for floating-point benchmarks: 30.5% for scenario I and 25% for scenario II. This in fact is due to the larger average occupancy of FP benchmarks compare to INT benchmarks as presented in Table I. As expected, for most benchmarks, the ROB occupancy is smaller for scenario II compared to I. This is consistent with the results in Fig. 1: the reduction in issue rate results in the ROB occupancy increase. Table II reports the relative load queue (LQ)

TABLE I
PERCENT ROB OCCUPANCY INCREASE DURING CACHE MISS PERIOD RELATIVE TO NORMAL PERIOD

***ROB* occupancy increase**	**Scenario I**	**Scenario II**	**Average Occupancy**		**Scenario I**	**Scenario II**	**Average occupancy**
bzip2	165.0	88.6	40	applu	13.8	-4.9	83
Crafty	179.6	63.6	38	apsi	46.6	18.2	73
Gap	16.6	61.7	55	Art	31.7	56.9	68
Gcc	97.7	43.9	42	equake	49.8	38.1	68
Gzip	152.9	41.0	34	facerec	87.9	14.1	74
Mcf	42.2	40.6	60	galgel	30.9	34.4	82
Parser	31.3	102.3	40	lucas	-0.7	54.0	66
Twolf	81.8	58.8	42	mgrid	8.8	5.6	91
Vortex	118.7	57.8	52	swim	-4.3	11.4	94
Vpr	96.6	55.7	61	wupwise	40.2	24.4	76
INT average	***98.2***	***61.4***	46	***FP average***	***30.5***	***25.2***	78

TABLE II
PERCENT LQ OCCUPANCY INCREASE DURING CACHE MISS PERIOD RELATIVE TO NORMAL PERIOD

***LQ* occupancy increase**	**Scenario I**	**Scenario II**		**Scenario I**	**Scenario II**
bzip2	69.6	32.7	applu	17.3	25.7
Crafty	111.8	24.2	apsi	38.8	18.5
Gap	27.3	36.7	Art	4.0	29.1
Gcc	52.1	27.6	equake	33.1	41.5
Gzip	70.2	22.9	facerec	68.6	72.1
Mcf	24.6	21.5	galgel	43.0	8.9
Parser	54.8	17.9	lucas	19.1	81.8
Twolf	128.4	171.2	mgrid	25.3	13.9
Vortex	88.8	66.3	swim	4.3	20.3
Vpr	127.6	54.5	wupwise	28.5	32.8
INT average	75.5	47.6	***FP average***	28.2	34.5

TABLE III
PERCENT SQ OCCUPANCY INCREASE DURING CACHE MISS PERIOD RELATIVE TO NORMAL PERIOD

***SQ* occupancy increase**	**Scenario I**	**Scenario II**		**Scenario I**	**Scenario II**
bzip2	50.3	26.4	applu	28.3	16.5
Crafty	42.5	16.3	apsi	49.8	29.4
Gap	79.5	44.1	Art	15.7	19.6
Gcc	19.4	22.9	equake	22.1	14.2
Gzip	36.4	25.1	facerec	44.2	23.8
Mcf	47.1	63.5	galgel	27.5	35.4
Parser	11.8	21.5	lucas	6.1	19.4
Twolf	37.4	29.2	mgrid	34.7	20.4
Vortex	63.2	29.6	swim	11.5	7.1
Vpr	33.8	42.6	wupwise	14.9	11.6
INT average	42.1	32.2	***FP average***	25.5	19.7

occupancy increase during cache miss period. The occupancy increase is large during both scenarios. We also present the occupancy increase of store queue (SQ) during cache miss period in Table III. The occupancy increases noticeably but at a lower rate compare to load queue. In most benchmarks during scenario I where an L2 miss is pending, both LQ and SQ occupancy increase more compare to scenario II where multiple L1 misses are pending. Similar behavior was observed for ROB. A larger miss penalty of L2 cache compare to L1 data cache could explain such occupancy difference.

TABLE IV
AVERAGE INTEGER REGISTER FILE OCCUPANCY DURING CACHE MISS PERIOD (SCENARIOS I AND II) AND NORMAL PERIOD

Register File occupancy	**Scenario I *IRF***	**non-Scenario I *IRF***	**Scenario II *IRF***	**non-Scenario II *IRF***
bzip2	74.4	28.8	56.6	30.7
crafty	83.4	31.9	51.4	32.2
gap	46.2	41.1	65.8	42.9
gcc	46.3	21.2	28.7	24.0
gzip	45.1	27.2	39.8	27.2
mcf	40.8	29.3	46.8	36.4
parser	37.4	29.8	57.0	29.8
twolf	58.7	32.3	46.0	29.8
vortex	70.9	31.1	52.4	35.0
vpr	63.9	29.0	66.4	41.0
INT average	**55.3**	**29.2**	**50.3**	**32.0**
applu	6.0	5.6	1.7	6.2
apsi	16.1	18.3	15.8	17.9
art	35.4	25.0	23.0	29.0
equake	34.2	27.4	32.7	29.4
facerec	52.6	22.5	30.3	38.4
galgel	50.4	27.4	32.1	26.0
lucas	21.7	23.8	41.7	22.1
mgrid	5.9	6.2	1.9	6.4
swim	23.3	27.8	29.7	23.1
wupwise	26.3	28.8	40.5	26.9
FP average	**26.6**	**20.9**	**24.0**	**22.1**

We also present the average IRF and FRF occupancies for scenarios I and II and also the normal period (see Tables IV and V). As the results show, 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.

TABLE V
AVERAGE FLOATING POINT REGISTER FILE OCCUPANCY DURING CACHE MISS PERIOD (SCENARIOS I AND II) AND NORMAL PERIOD

Register File occupancy	Scenario I *FRF*	non-Scenario I *FRF*	Scenario II *FRF*	non-Scenario II *FRF*
bzip2	0.0	0.0	0.0	0.0
crafty	0.1	0.0	0.0	0.0
gap	0.1	0.7	0.6	0.5
gcc	0.2	0.1	0.0	0.1
gzip	0.0	0.0	0.0	0.0
mcf	1.0	1.1	3.2	0.1
parser	0.0	0.0	0.1	0.0
twolf	2.6	2.1	2.5	2.0
vortex	0.3	0.2	0.2	0.2
vpr	7.8	8.6	8.7	8.3
INT average	**1.1**	**1.2**	**1.4**	**1.0**
applu	76.6	64.8	77.3	73.7
apsi	65.7	37.6	58.8	43.6
art	36.2	30.7	42.9	6.3
equake	16.1	7.1	21.0	9.6
facerec	50.0	28.9	48.1	35.0
galgel	41.8	48.7	61.0	44.2
lucas	47.7	44.0	29.7	47.0
mgrid	90.0	80.7	96.7	87.2
swim	77.1	78.1	87.1	76.2
wupwise	53.5	28.7	38.0	42.2
FP average	**56.5**	**44.7**	**56.2**	**46.0**

Based on the results shown, in the next section we propose a simple algorithm to reduce the power dissipation in the ROB, LQ, SQ, the register files, and the issue/wakeup units.

IV. PROPOSED CENTRALIZED APPROACH

The approach proposed in this paper aims at reducing the dynamic and static power dissipation of the ROB, LQ, SQ, the register files, and the instruction queue, by adaptive resizing. Reducing the size of each of these units will require different hardware modifications (at extra power/area cost) which are determined by the resizing scheme. In order to minimize the hardware cost and complexity, we propose a simple resizing scheme that goes 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 Fig. 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, LQ, *and* SQ occupancies during cache miss periods, we propose to increase their size during such periods (during both scenarios I and II). During normal periods, we keep the ROB, LQ, *and* SQ size at half its possible size.

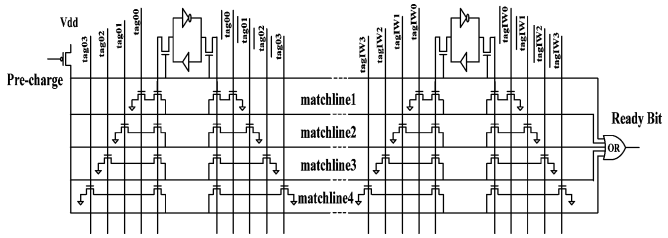


Fig. 2. Circuit implementation of instruction queue.

The dynamic adaptation of register file requires more caution since the FRF and IRF behave differently for integer and floating-point benchmarks. Based on the above discussion 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, 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. To distinct the integer and floating point benchmarks we can use the floating point register file occupancy. A small floating point register file occupancy, for instance below 10 entries on average, is an indication of an integer benchmark (results in Tables IV and V).

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

Cache miss period ends if one of the two following conditions met:

- the L2 cache miss is either serviced or flushed (part of a misprediction path);
- all pending DL1 misses are either serviced or flushed.

Most processors have exclusive registers to monitor the state of cache misses. In a case where these registers are not available, we need a 2-bit saturating counter for keeping the number of pending DL1 misses and a 1-bit registers for keeping the L2 miss.

In the next section we explain how to detect whether the augmented part is empty for each of ROB, LQ, SQ, and register file.

A. Reducing the Effective Width of Issue and Wakeup

We propose reducing the size of the wakeup and issue width from 4 to 2. Fig. 2 shows the circuit level implementation for one row of the instruction queue. At each cycle, the match lines are precharged 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 four instructions broadcasted on the taglines, we need to have four sets of one-bit comparators for each one-bit cell, as shown in Fig. 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 an instruction source operand showing that it is ready. Using spice measurements of a 32-entry IQ layout

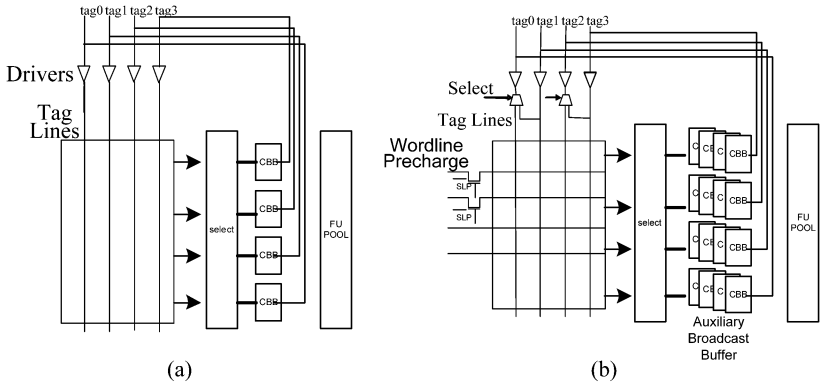


Fig. 3. (a) Baseline issue/wake up logic and (b) modified issue/wake up logic.

our results in accordance with [14], [22], 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. Precharging 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 32×4 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.

Results in Fig. 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 precharging the other matchlines is not useful. We can thus prevent precharging such matchlines by using a gated-Vdd transistor as shown in Fig. 3(b). Therefore, the question is how to determine which set of matchlines associated with specific tagline should be disabled rather than precharged. Note that when the precharge line is disabled, the matchline is forced to drive "0" to the OR logic to avoid setting the ready bit.

The matchlines are being precharged when the clock is low and are conditionally discharged immediately after the results have been broadcasted on the taglines (when the clock is high). Precharging 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 would allow us to disable them. In Fig. 3(b), we show the circuit modification needed to reduce the wakeup width from 4 to 2 and hence the disabling/precharging 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. Note that the delay of the demultiplexer used is only few pico seconds, far less than the operating clock period (500 ps). We assume that such a delay does not impact the processor operating clock frequency.

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 (referring as auxiliary broadcast buffer) half of the tags and broadcast them whenever an active becomes available. As explained in [54], to wakeup a dependent instruction, the scheduler uses a countdown timer which is initialized with the instruction latency. Once the counter reaches one, the corresponding buffer broadcasts tags to all instructions in the instruction queue. In a case where the counter reaches one but the tagline is not available the tags are buffered and will be broadcasted as soon as the corresponding tagline 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 four-entry broadcast buffer is sufficient to minimize such occurrence. In the very rare case that the broadcast buffer fills up, the issue stage 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.

B. Dynamically Resizing ROB and Register Files

Fig. 4 shows the circuit level implementation of an SRAM. To read or write an entry each cycle all the bit lines must be precharged 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 $\overline{\text{bitline}}$ 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

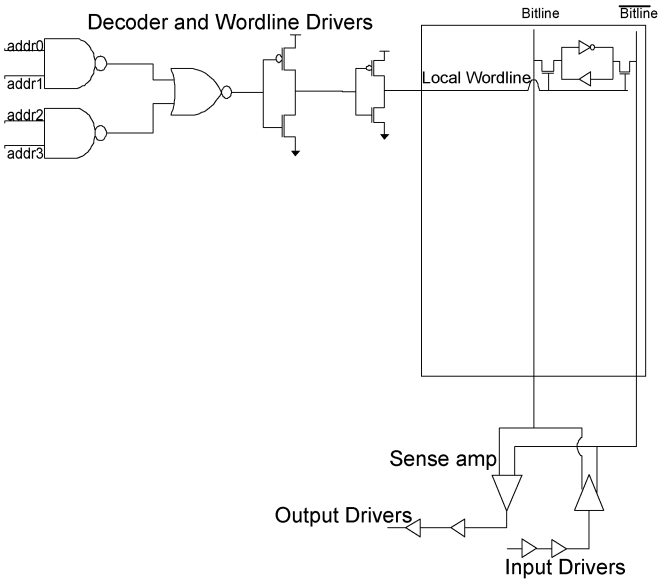


Fig. 4. ROB SRAM circuit.

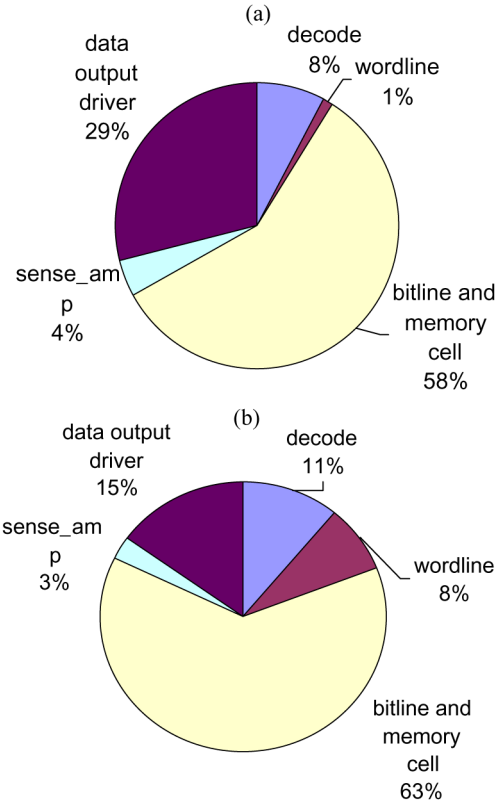


Fig. 5. (a) Dynamic energy and (b) leakage power of the register file.

to W entries in the same cycle, we must have $(N+W)$ bit lines (register file is single ended). 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. Precharging and discharging the large number of bit lines are the major sources of power dissipation in these two structures [16], [28].

Using a modified version of CACTI4 which models a single-ended multiported SRAM [15] in Fig. 5 we report 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, prechargers, and input and output drivers as explained in [16]. Reference [16] proposed to partition the ROB into eight units. This requires eight times more sense amps precharge 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.

As shown in Fig. 6, two or more SRAM cells are combined together to divide the bit line into several sub-bit lines. In the nondivided 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 Fig. 6 as segment control switch). As a side effect, the large number of segments increases the area and power overhead.

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

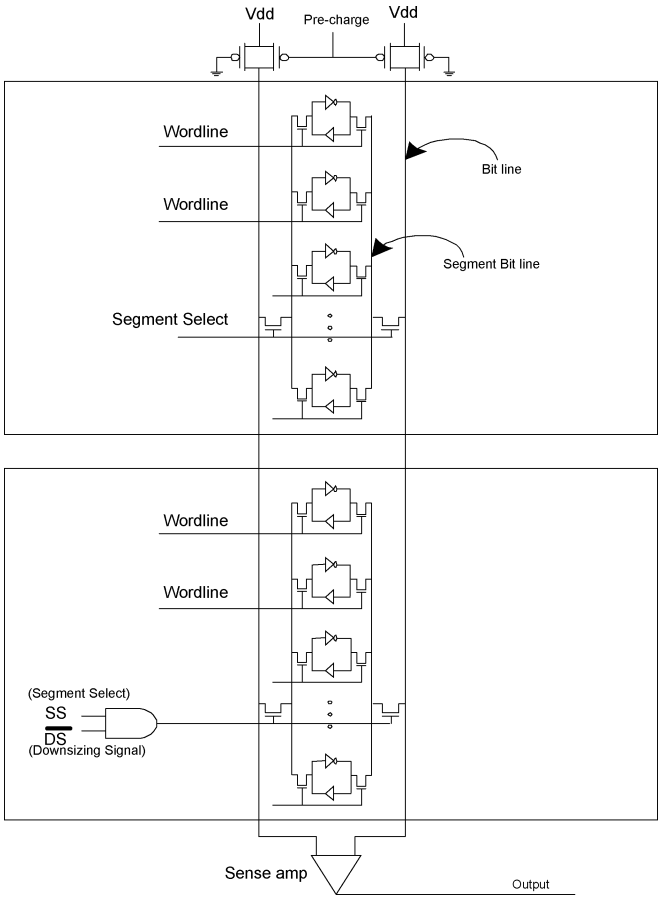


Fig. 6. Divided bit line circuit.

by deasserting the downsize signal and turning on the disabled partition. The overhead of downsizing and upsizing is one cycle (gated-Vdd overhead). The end of a 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 approach applied to ROB can also be 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). By ORing these bits we can detect when the partition is empty. Note that the logic to OR all bits in the lower partition is not on the critical path, as the downsize decision is done in parallel to accessing ROB and register file. We assume that the additional bit in each row (entry) of the lower partition does not increase its access delay beyond the clock period.

C. Dynamically Resizing Load and Store Queue

LQ (SQ) is basically a CAM+RAM structure which has a similar circuit implementation to the instruction queue (shown in Fig. 2). Our result indicates that matchline discharge and tagline broadcast are the major energy consumption activity in a 32×64 bits LQ and SQ. This is in agreement with the results reported in [38], [40]–[44]. Specifically the tagline broadcast is responsible for almost 36% of energy consumption in the LQ and SQ during an associative search operation. As the tagline must go across the entire height of the LQ and SQ, it has a large wire capacitance. It also has to drive a large number of one-bit comparators. The equivalent tagline capacitance is equal to

$$\begin{aligned} \mathrm{C_{tagline}} &= \mathrm{C_{gate}(CompareEn)} * \mathrm{N_{rows}} \\ &\quad + \mathrm{C_{diff}(Compare\ Driver)} + \mathrm{C_{metal}} * \mathrm{T_{lenghth}} \end{aligned}$$

where $\mathrm{N_{rows}}$ is the number of CAM rows and $\mathrm{T_{lenghth}}$ is the length of a tagline. Driving this large capacitor is responsible for a large portion of power consumption in the LQ and SQ. To reduce the power dissipation of the LQ and SQ one can dynamically reduce their effective size; N_{rows} which reduces the equivalent gate capacitance of the taglines. The circuit modification to dynamically adapt the size of LQ (and SQ) is shown in Fig. 7. For this purpose, the LQ (and SQ) is divided into two segments of 16 entries each which are connected through transmission gates as shown in Fig. 7. This allows the upper segment tagline to become isolated from the lower segment tagline if the pass gate is off. During the *normal period* the lower segment is power gated and the transmission gate is turned off to isolate the lower bitline segment from the upper bitline segment. During this period the tags are only broadcasted to the the upper segment entries. Since the lower tagline segment is floating, the equivalent tagline capacitance during normal period is decided by the upper segment tagline. Therefore, the tagline delay in this case remains close to the tagline delay of a 16-row LQ (and SQ). The only difference is the delay added by the source capacitance of the pass gate, which is negligible.

In addition, we need to be able to detect when the lower segment is empty (for downsizing at the end of cache miss period when the added segment is empty). To do that we have augmented the lower segment with one extra bit per entry. This bit is set when an LQ (and SQ) entry is taken and is being reset when the entry is released. By ORing these bits we can detect when the segment is empty. Similar to ROB and RF, the logic to OR all bits in the lower partition of LQ (and SQ) is not on the critical path, as the downsize decision is done in parallel to accessing LQ (and SQ). We also assume that the additional bit in each row of the lower partition does not increase its access delay beyond the clock period. Since the remaining components of LQ delay change very slightly, the LQ access delay remains close to that for a 16-row LQ when the upper segment is isolated (the same hold true for SQ). Once the LQ (and SQ) is downsized we will safely power gate all entries in the lower segment to reduce their leakage power.

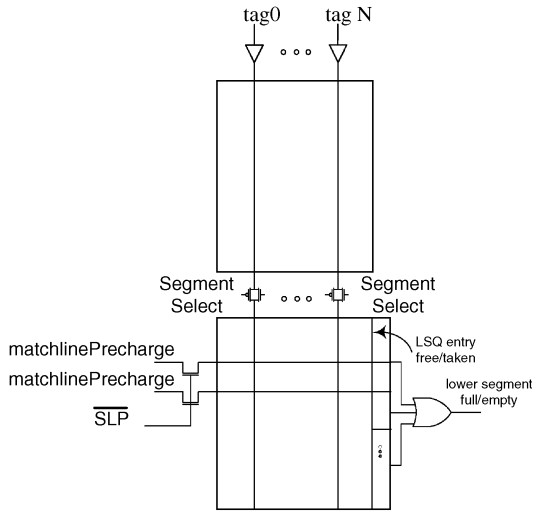


Fig. 7. LQ (and SQ) implementation for dynamic adaptation.

TABLE VI
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	"tournament" predictor combining a 8192 entries bimodal and 8192 entries 2-level global predictor with 12 bits history width,4K-entry BTB	**Complex unit**	2 INT, 2 FP multiply/divide units
Reorder buffer	128 entry	**Pipeline**	15 cycles (some stages are multi-cycles)

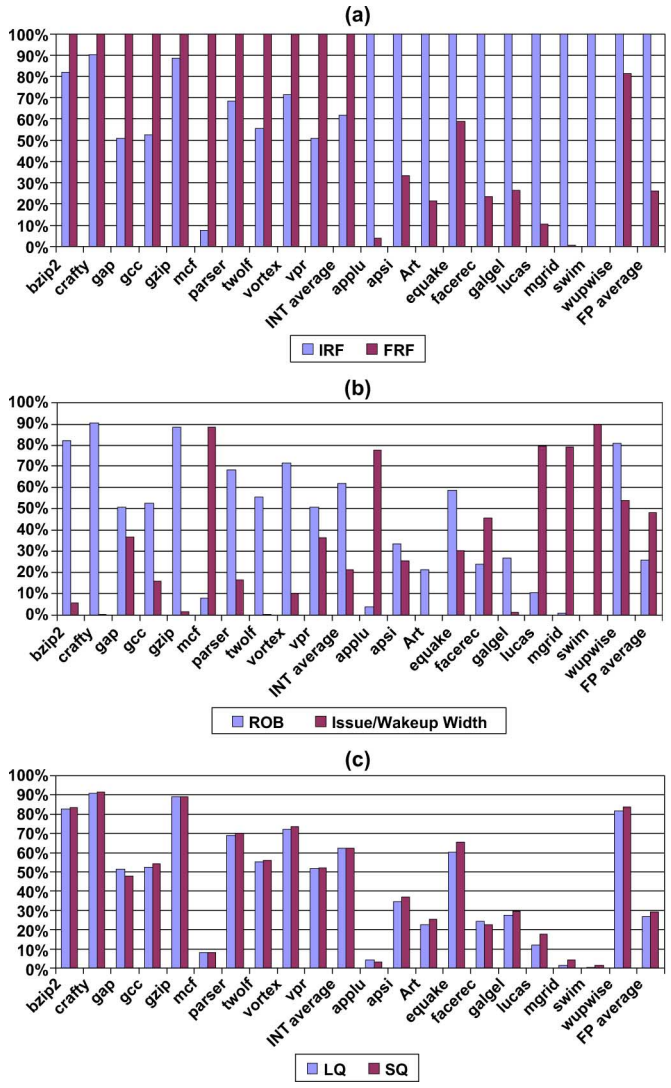


Fig. 8. Percent time with size reduction for: (a) IRF, FRF; (b) ROB and issue/wakeup width; and (c) LQ and SQ.

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 VI describes the processor architecture, the clock frequency is 2 GHz. SPEC2K benchmarks were compiled with the O4 flag using the Compaq compiler for the Alpha 21264 processor and executed with reference data sets.

The architecture was simulated using an extensively modified version of SimpleScalar 4.0 [23]. The benchmarks were fast-forwarded for two billion instructions, then fully simulated for two billion instructions. A modified version of CACTI4 [15] was used for estimating power in the ROB and the Register files in 65-nm technology. For estimation of energy in IQ, LQ, and SQ we used spice simulation on the actual VLSI layouts of the IQ, LQ, and SQ. The power in the Instruction Queue, LQ, and SQ was evaluated using SPICE and the TSMC 65-nm technology with Vdd at 1.08 V.

VI. RESULTS

Power savings and performance changes associated with our approach are shown in Figs. 8–10. The approach is used for the ROB, register files, LQ, SQ, and issue/wakeup width simultaneously. Fig. 8 shows the fraction of execution time when the ROB, IRF, and FRF, LQ, SQ, and issue/wakeup width were reduced to half their size for each benchmark.

Table VII shows cache miss rates for DL1 and L2 caches. Larger cache misses result in more frequent unit upsizing and therefore a smaller fraction of execution time where units can be downsized to half of their original size. The results in Fig. 8 correlate well with the results in Table VII. In floating point benchmarks compare to integer benchmarks, due to a large DL1 and L2 cache miss rates, there is less opportunity to downsize RF, ROB, LQ, and SQ. Compare to other floating point benchmarks, in equake and wupwise there is more opportunity to downsize units. In fact in these two benchmarks in spite of large L2 miss rate, a small DL1 miss rate results in overall smaller number of cache misses (DL1 and L2) and therefore there is more opportunity to keep any of RF, ROB, LQ, and SQ downsized. IPC reduction is shown in Fig. 9. Leakage and dynamic power reduction for individual units is shown in Fig. 10. On average, the performance loss is 1.1% for integer benchmarks and 2.4% for

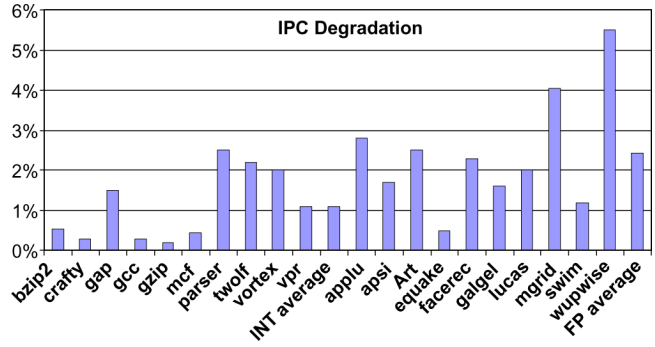


Fig. 9. IPC degradation due to *resource resizing*.

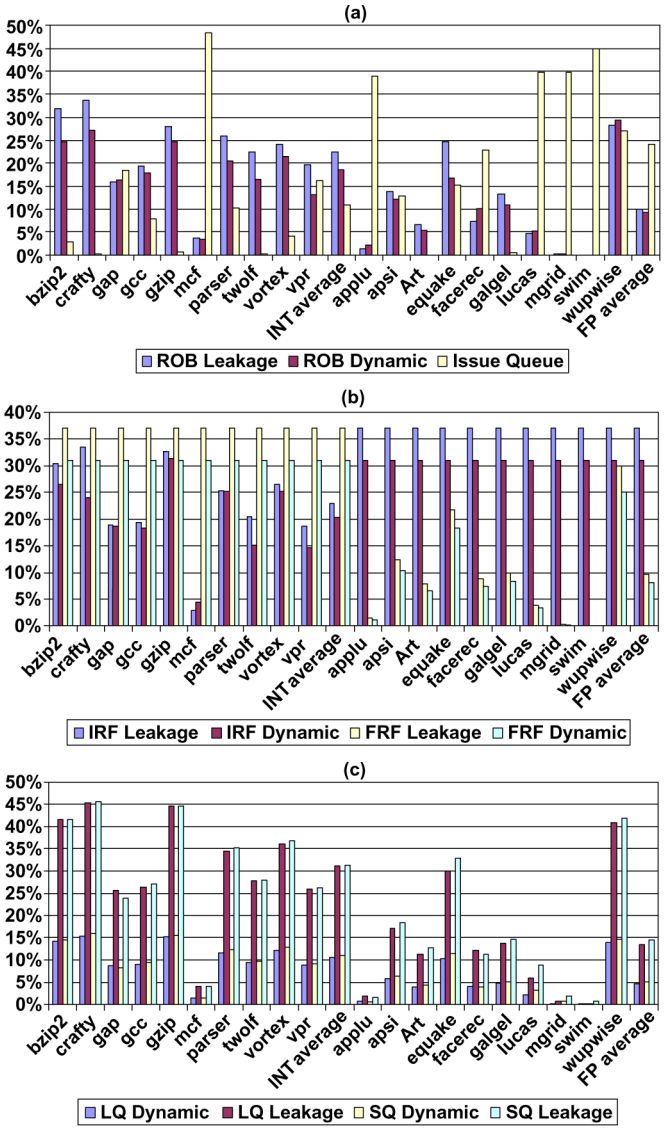


Fig. 10. Dynamic and leakage power reduction in: (a) ROB and instruction queue; (b) IRF and FRF; (c) LQ and SQ.

floating-point benchmarks. In wupwise and mgird we observe the highest performance degradation. To explain the large performance degradation in wupwise we refer to results presented earlier in Fig. 8. As shown, in wupwise FRF, ROB, LQ, and SQ spends a large portion of the execution time downsized. Using

TABLE VII
CACHE MISS RATES FOR DL1 AND L2 CACHE

Cache miss rate	DL1	L2		DL1	L2
bzip2	0.0167	0.0417	applu	0.0563	0.6572
Crafty	0.0021	0.0087	apsi	0.0273	0.2778
Gap	0.0069	0.5506	Art	0.4141	0.0001
Gcc	0.0455	0.0366	equake	0.0172	0.6727
Gzip	0.0067	0.0468	facerec	0.0340	0.3121
Mcf	0.2385	0.4284	galgel	0.0366	0.0057
Parser	0.0197	0.0688	lucas	0.0966	0.6657
Twolf	0.0536	0.0003	mgrid	0.0356	0.4587
Vortex	0.0027	0.2314	swim	0.0893	0.6308
Vpr	0.0231	0.1476	wupwise	0.0124	0.6740
INT average	***0.04155***	***0.15609***	***FP average***	***0.08194***	***0.43548***

a smaller size of register file, reorder buffer and load and store queue increases the chances of processor stalls which in turn impacts performance noticeably. For mgrid the story is different as FRF, ROB, LQ, and SQ spend only a very small portion of execution time downsized. In fact in mgrid the reorder buffer and floating point register file is almost full most the time (having average occupancy of more than 90 entries as reported in Tables I and V). As reported in Table I, in mgrid there is a very small variation in size of ROB during scenario I and II. So In fact reducing the size of ROB results in the processor being stalled more frequently. This in fact is the reason of a large performance degradation in mgrid. To minimize the IPC reduction one solution is to monitor average occupancy run-time in all ROB, LQ, SQ, IRF, FRF, and IQ. IF the average occupancy is large (for instance more than half of the unit capacity), we can avoid applying resource resizing. Doing so would reduce the chances of processor being stalled because of unavailable resources and thus it impact performance less.

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 is a result of higher L2 miss rate in floating point benchmarks compared to integer benchmarks (note that the issue/wakeup width is reduced only during scenario I which is L2 miss period).

The same figure shows 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 Fig. 8).

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).

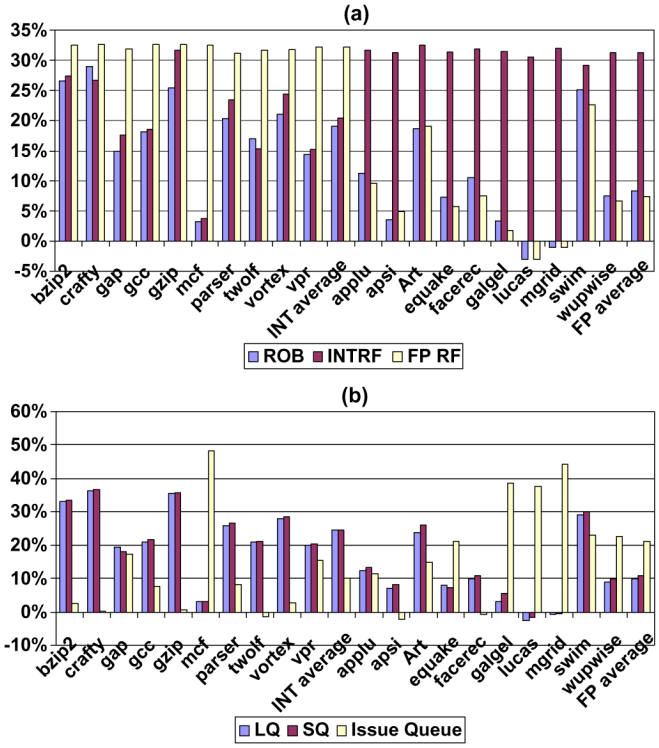


Fig. 11. Total energy-delay reduction. (a) ROB and register file. (b) LSQ and issue queue.

Results for LQ and SQ are presented in Figs. 8 and 10. In integer benchmarks LQ and SQ power reduce more noticeably compare to floating point benchmarks. This is due to the fact that in integer benchmark there is more opportunity for resizing as results in Fig. 8 show. The store queue power reduces slightly more than the load queue. Note that LQ/SQ can be downsized only when a cache miss period is over and when their lower segment is empty. The lower segment of SQ is becoming empty faster than the lower segment of LQ. Therefore the store queue is kept in low power mode slightly more than the load queue. The average dynamic power reduction in LQ and SQ is 10% and 11%, respectively, in integer benchmark. This is lower in floating point benchmark: almost 5% for both LQ and SQ.

The leakage power reduces noticeably; by as much as 45% in integer benchmark and 42% in floating point benchmark in both LQ and SQ.

Fig. 11 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. The increase in energy-delay product is in fact stems from either large performance degradation which makes the delay larger (for mgrid) or small energy improvement in present of some performance degradation (for lucas) which increases the overall energy and delay product.

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. As reported in Fig. 10 a large leakage and dynamic power reduction is observed for IRF which makes the overall energy delay product smaller across all benchmarks. For LQ this is the case for lucas and mgrid with 2.7% and 0.88% increase respectively. For the same benchmarks the energy-delay product increases slightly in SQ; in lucas by 1.7% and in mgrid by 0.56%. These are the benchmarks in which some noticeable performance degradation is observed; this is specifically the case for mgrid with almost 3.6% performance impact.

Overall the energy delay reduces more noticeably in integer benchmarks compare with floating point benchmarks. This is due to the fact that the performances degrade smaller/negligible in integer benchmarks compare to floating point benchmarks as shown in Fig. 9. On average the total energy-delay product decreases by 19%, 20%, 32%, 24%, 24%, and 10% for ROB, IRF, FRF, LQ, SQ, and instruction queue respectively in integer benchmarks. In floating point benchmarks this is 8%, 31%, 7%, 10%, 11%, and 21%, respectively. The overall average energy-delay product reduces by 15%, 26%, 20%, 17%, 18%, and 15%, respectively.

VII. Conclusion and Future Work

This paper presented a centralized mechanism to simultaneously reduce both leakage and dynamic power in all major on-core CAM and SRAM-based processor units; reorder buffer, register files, instruction queue, and load and store queue. It proposed to dynamically adjust the size of these units during cache miss periods. It also showed how to modify the circuitry for each of these units to apply this mechanism. The required hardware modifications are minimal and do not require significant redesign and verification efforts.

Applying the new algorithm, the total energy-delay product is reduced, on average, by 15%, 26%, 20%, 17%, 18%, and 15% for the reorder buffer, the integer register file, the floating-point register file, the instruction queue and the load and the store 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 on average, 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.

Acknowledgment

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 U.S. National Science Foundation.

References

[1] R. Gonzalez, A. Cristal, A. Veidenbaum, and M. Valero, "A content aware register file organization," in *Proc. 31st Int. Symp. Comput. Arch. (ISCA)*, 2004, p. 314.

[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 Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 1, pp. 18–28, Jan. 2005.

[3] M. A. Ramirez, A. Cristal, A. Veidenbaum, L. Villa, and M. Valero, "A new pointer-based instruction queue design and its power-performance evaluation," in *Proc. IEEE Int. Conf. Comput. Des. (ICCD)*, 2005, pp. 647–653.

[4] C. Hsu and W. Feng, "Effective dynamic voltage scaling through CPU-boundedness detection," in *Proc. 4th IEEE/ACM Workshop Power-Aware Comput. Syst.*, 2004, p. 135.

[5] H. Li, C.-Y. Cher, T. Vijaykumar, and K. Roy, "VSV: L2-miss-driven variable supply-voltage scaling for low power," in *Proc. Int. Symp. Microarch.*, 2003, pp. 19–28.

[6] E. Borch, E. Tune, S. Manne, and J. Emer, "Loose loops sink chips," in *Proc. 8th Int. Symp. High Perform. Comput. Arch.*, Feb. 2002, pp. 299–310.

[7] D. Balkan, J. Sharkey, D. Ponomarev, and A. Aggarwal, "Address-value decoupling for early register deallocation," in *Proc. 35th Int. Conf. Parallel Process. (ICPP)*, 2006, pp. 337–346.

[8] I. Park, M. D. Powell, and T. N. Vijaykumar, "Reducing register ports for higher speed and lower energy," presented at the Int. Symp. Microarch. (MICRO-35), Istanbul, Turkey, 2002.

[9] H. Homayoun, S. Pasricha, M. Makhzan, and A. Veidenbaum, "Improving performance and reducing energy-delay with adaptive resource resizing for out-of-order embedded processors," presented at the ACM SIGPLAN/SIGBED Conf. Lang., Compilers, Tools for Embed. Syst. (LCTES), Tucson, AZ, 2008.

[10] M. Fleischmann, "Crusoe power management: Cutting x86 operating power through longrun," in *Embed. Processor Forum*, 2000, p. 195.

[11] J. H. Tseng and K. Asanović, "Banked multiported register files for high-frequency superscalar microprocessors," in *Proc. 30th Int. Symp. Comput. Arch.*, 2003, pp. 62–71.

[12] H. Homayoun and A. Baniasadi, "Using lazy instruction prediction to reduce processor wakeup power dissipation," presented at the 2nd Workshop Unique Chips Syst., in Conjunction With IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS), Austin, TX, 2006.

[13] E. Tune, R. Kumar, D. M. Tullsen, and B. Calder, "Balanced multithreading: Increasing throughput via a low cost multithreading hierarchy," in *Proc. 37th Annu. Int. Symp. Microarch. (MICRO-37)*, Portland, OR, 2004, pp. 183–194.

[14] A. Buyuktosunoglu, D. H. Albonesi, P. Bose, P. W. Cook, and S. E. Schuster, "Tradeoffs in power-efficient issue queue design," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2002, pp. 184–189.

[15] Hewlett-Packard Company, Palo Alto, CA, "CACTI4," [Online]. Available: http://quid.hpl.hp.com:9081/cacti/

[16] D. Ponomarev, G. Kucuk, and K. Ghose, "Energy-efficient design of the reorder buffer," presented at the Int. Workshop Power Tim. Model., Opt. Simulation (PATMOS), Seville, Spain, 2002.

[17] J. L. Ayala, M. Lopez-Vallejo, A. Veidenbaum, and C. A. Lopez, "Energy aware register file implementation through instruction predecode," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch., Processors (ASIP)*, 2003, pp. 86–96.

[18] D. Ponomarev, G. Kucuk, and K. Ghose, "Dynamic allocation of datapath resources for low power," presented at the Workshop Complexity-Effective Des. (WCED), Göteborg, Sweden, 2001.

[19] H. Homayoun, S. Pasricha, M. Makhzan, and A. Veidenbaum, "Dynamic register file resizing and frequency scaling to improve embedded processor performance and energy-delay efficiency," presented at the 45th Des. Autom. Conf., Anaheim, CA, 2008.

[20] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster, "Dynamically tuning processor resources with adaptive processing," *IEEE Comput.*, vol. 36, no. 12, pp. 49–58, Dec. 2003.

[21] I. Bahar and S. Manne, "Power and energy reduction via pipeline balancing," in *Proc. Int. Symp. Comput. Arch. (ISCA)*, 2001, pp. 218–229.

[22] R. Canal and A. Gonzalez, "Reducing the complexity of the issue logic," presented at the Int. Conf. Supercomput., Naples, Italy, 2001.

[23] SimpleScalar LLC, "SimpleScalar4 Tutorial," [Online]. Available: 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," presented at the Int. Symp. Microarch., Austin, TX, 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," presented at the IEEE Int. Symp. Low Power Electron. Des., Rapallo, Italy, 2000.

[26] A. Karandikar and K. K. Parhi, "Low power SRAM design using hierarchical divided bit-line approach," in *Proc. Int. Conf. Comput. Des.*, 1998, pp. 82–88.

[27] M. Huang, J. Renau, and J. Torrellas, "Energy-efficient hybrid wakeup logic," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2002, pp. 196–201.

[28] V. V. Zyuban and P. M. Kogge, "The energy complexity of register files," presented at the Int. Symp. Low Power Electron. Des., Monterey, CA, 1998.

[29] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," presented at the 34th IEEE/ACM Int. Symp. Microarch. (MICRO-34), Austin, TX, 2001.

[30] J. S. Hu, N. Vijaykrishnan, and M. J. Irwin, "Exploring wakeup-free instruction scheduling," presented at the 10th Int. Conf. High-Perform. Comput. Arch. (HPCA-10), Madrid, Spain, 2004.

[31] J. Alastruey, T. Monreal, V. Viñals, and M. Valero, "Microarchitectural support for speculative register renaming," presented at the Proc. 21st IEEE Int. Parallel Distrib. Process. Symp., Long Beach, CA, 2007, "," in .

[32] D. Balkan, J. Sharkey, D. Ponomarev, and K. Ghose, "SPARTAN, speculative avoidance of register allocation to transient values for performance and energy efficiency," in *Proc. 15th Int. Conf. Parallel Arch. Compilation Techn. (PACT)*, 2006, pp. 265–274.

[33] D. Folegnani and A. González, "Energy-effective issue logic," presented at the 28th Annu. Int. Symp. Comput. Arch., Göteborg, Sweden, May 2001.

[34] J. Abella, R. Canal, and A. González, "Power- and complexity-aware issue queue designs," *IEEE Micro*, vol. 23, no. 5, pp. 50–58, Sep.–Oct. 2003.

[35] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," presented at the 24th Annu. Int. Symp. Comput. Arch., Denver, CO, 1997.

[36] D. Ernst, A. Hamel, and T. Austin, "Cyclone: A broadcastfree dynamic instruction scheduler selective replay," presented at the 30th Annu. Int. Symp. Comput. Arch., San Diego, CA, 2003.

[37] R. Canal and A. Gonzalez, "Reducing the complexity of the issue logic," presented at the Int. Conf. Supercomput., Naples, Italy, 2001.

[38] I. Park, C. L. Ooi, and T. N. Vijaykumar, "Reducing design complexity of the load/store queue," presented at the Int. Symp. Microarch., San Diego, CA, 2003.

[39] J. Abella and A. González, "SAMIE-LSQ: Set-associative multiple-instruction entry load/store queue," presented at the IEEE Int. Parall. Distrib. Process. Symp. (IPDPS), Rhodes Island, Greece, 2006.

[40] T. Sha, M. Martin, and A. Roth, "NoSQ: Store-load communication without a store queue," presented at the Int. Symp. Microarch. (Micro), Chicago, IL, 2007.

[41] Y. Tsai, C. J. Hsu, and C. H. Chen, "Power-efficient and scalable load/store queue design via address compression," presented at the ACM Symp. Appl. Comput., Ceara, Brazil, 2008.

[42] F. Castro, D. Chaver, L. Pinuel, M. Prieto, M. C. Huang, and F. Tirado, "A power-efficient and scalable load-store queue design," presented at the Int. Workshop Power Tim. Model. Opt. Simulation (PATMOS), Leuven, Belgium, 2005.

[43] J. Sharkey, K. Ghose, D. V. Ponomarev, and O. Ergin, "Power-efficient wakeup tag broadcast," presented at the Int. Conf. Comput. Des. (ICCD), San Jose, CA, 2005.

[44] A. Buyuktosunoglu, D. Albonesi, S. Schuster, and D. Brooks, "A circuit level implementation of an adaptive issue queue for power-aware microprocessors," presented at the Great Lakes Symp. VLSI Syst. (GLSVLSI), West Lafayette, IN, 2001.

[45] Y. Han, I. Koren, and C. A. Moritz, "Temperature aware floorplanning," presented at the Workshop Temp. Aware Comput. Syst., Madison, WI, 2005.

[46] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau, "Power model validation through thermal measurements," presented at the Int. Symp. Comput. Arch., San Diego, CA, 2007.

[47] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," presented at the ISCA, San Diego, CA, 2003.

[48] H. Homayoun, M. Makhzan, J. L. Gaudiot, and A. Veidenbaum, "A centralized cache miss driven technique to improve processor power dissipation," presented at the Int. Symp. Syst., Arch., Model. Simulation (SAMOS VIII), Samos, Greece, 2008.

[49] J. Hasan, A. Jalote, T. Vijaykumar, and C. Brodley, "Heat stroke: Power-density-based denial of service in SMT," presented at the Int. Symp. High-Perform. Comput. Arch., San Francisco, CA, 2005.

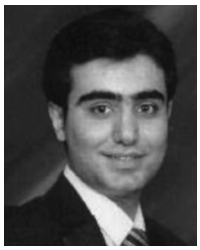
[50] Intel Corporation, Santa Clara, CA, "Intel," [Online]. Available: http://www.intel.com/technology/architecture-silicon/next-gen/

[51] A. Aggarwal, M. Franklin, and O. Ergin, "Defining wakeup width for efficient dynamic scheduling," presented at the Int. Conf. Comput. Des. (ICCD), San Jose, CA, 2004.

[52] TaliMoreshet and R. Iris Bahar, "Power-aware is-sue queue design for speculative instructions," in *Proc. 40th Conf. Des. Autom. (DAC)*, New York, 2003, pp. 634–637.

[53] J. Carretero, P. Chaparro, X. Vera, J. Abella, and A. González, "End-to-end register data-flow continuous self-test," presented at the Int. Symp. Comput. Arch. (ISCA), Austin, TX, 2009.

[54] M. D. Brown, J. Stark, and Y. N. Patt, "Select-free instruction scheduling logic," in *Proc. 34th Int. Symp. Microarch.*, 2001, pp. 204–213.



Houman Homayoun (M'10) received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2003, the M.S. degree in computer engineering from University of Victoria, Victoria, BC, Canada, in 2005, and the Ph.D. degree in computer science from the University of California, Irvine (UC-Irvine), in 2010.

His research is on power-temperature and reliability-aware memory and processor design optimizations and spans the areas of computer architecture, circuit design and VLSI-CAD, where he has published over 30 technical papers on the subject. His research is among the first in the field to address the importance of cross-layer power and temperature optimizations in SRAM memory peripheral circuits.

Dr. Homayoun was named a 2010 National Science Foundation Computing Innovation Fellow by the Computing Research Association (CRA) and the Computing Community Consortium (CCC). He was a recipient of the four-years UC-Irvine computer science department chair fellowship.

Avesta Sasan (S'05) received the B.S. degree (*summa cum laude*) in computer engineering, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Irvine, in 2005, 2006, and 2010, respectively.

His research interests include low power design, process variation aware architectures, fault tolerant computing systems, nano-electronic power and device modeling, VLSI signal processing, processor power and reliability optimization and logic-architecture-device co-design. His latest publication, research outcomes, and updates can be found on http://www.avestasasan.com.

Jean-Luc Gaudiot (F'99) received the diplôme d'Ingénieur from the École Supeérieure d'Ingénieurs en Electrotechnique et Electronique, Paris, France, in 1976, and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1977 and 1982, respectively.

He is currently a Professor and a chair of the Electrical and Computer Engineering Department, the University of California, Irvine (UCI). Prior to joining UCI in January 2002, he was a Professor of electrical engineering with the University of Southern California since 1982, where he served as a Director of the Computer Engineering Division for three years. He has also worked on microprocessor systems design with Teledyne Controls, Santa Monica, CA (1979–1980), and research in innovative architectures with the TRW Technology Research Center, El Segundo, CA (1980–1982). He consults for a number of companies involved in the design of high-performance computer architectures. His research interests include multithreaded architectures, fault-tolerant multiprocessors, and implementation of reconfigurable architectures.

From 2006 to 2009, Dr. Gaudiot was the first Editor-In-Chief of the *IEEE Computer Architecture Letters*, a new publication of the IEEE Computer Society, which he helped found to the end of facilitating short, fast turnaround of fundamental ideas in the Computer Architecture domain. From 1999 to 2002, he was the Editor-In-Chief of the IEEE TRANSACTIONS ON COMPUTERS. In June 2001, he was elected chair of the IEEE Technical Committee on Computer Architecture, and re-elected in June 2003 for a second two-year term. He is a member of the ACM and the ACM SIGARCH. He has also chaired the IFIP Working Group 10.3 (Concurrent Systems). He was elevated to the rank of an AAAS fellow in 2007.

Alex Veidenbaum (M'08) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana-Champaign, in 1985.

He is currently a Professor with the Department of Computer Science, University of California, Irvine. His research interests include computer architecture for parallel, high-performance, embedded and low-power systems and compilers.

Prof. Veidenbaum is a member of the IEEE Computer Society and the ACM.