Turkish Journal of Electrical Engineering and Computer Sciences

Volume 20 | Number 1 Article 10

1-1-2012

Energy savings in simultaneous multi-threaded processors through dynamic resizing of datapath resources

GÜRHAN KÜÇÜK

MİNE MESTA

TÜBİTAK

Turk J Elec Eng & Comp Sci, Vol.20, No.1, 2012, © TÜBİTAK
doi:10.3906/elk-0909-233

Energy savings in simultaneous multi-threaded processors through dynamic resizing of datapath resources

Gürhan KÜÇÜK*, Mine MESTA
Department of Computer Engineering, Yeditepe University, Ataşehir, İstanbul-TURKEY
e-mails: gkucuk@cse.yeditepe.edu.tr, mmesta@cse.yeditepe.edu.tr

Received: 29.09.2009

Abstract

Nowadays, all the designers of systems from high-performance servers to battery-operated handheld devices aim for reliability, high-performance and longevity. Central within these aims the issue of processor power consumption is becoming increasingly important. In this study, we aim to adapt our already-proven method for single-threaded superscalar processors to simultaneous multi-threaded (SMT) processors for energy savings. The original method focused on resizing datapath resources according to the demands of running applications. To achieve this, the targeted resources are physically divided into multiple partitions, and turned on and off according to the needs of the applications. Since, the energy consumption of the turned-off datapath resources is quite low, as a result, it becomes possible to have great amount of energy savings within a processor. However, special care must be taken when there are multiple threads racing against each other to gain access to shared datapath resources. As a result, our proposed microarchitectural technique achieves 0.5% Instructions Per Cycle (IPC) and 3.2% Total number of instructions Per Cycle (TPC) improvement, while it turns off 45% of the Reorder Buffer (ROB), 59% of the Load-Store Queue (LSQ), 43% of the Issue Queue (IQ), 30% of the integer Physical Register Files (PRF) and, finally, 48% of the floating PRF, on the average across all simulated benchmarks. According to our estimates, the total processor power is reduced by 12%, on the average.

Key Words: *Microarchitectural techniques, energy reduction, simultaneous multi-threaded processors*

1. Introduction

Processors get faster and more complex, becoming more and more energy-hungry. In [1], it is mentioned that power densities of new generation processors approach the power densities of nuclear power plants. Lacking

*Corresponding author: Department of Computer Engineering, Yeditepe University, Ataşehir, İstanbul-TURKEY
This research has been funded by TÜBİTAK under grant no: 107E196.

intervention, the heat that comes out from the processors, working with this power density, would give rise to mechanical and electrical problems. The famous Arrhenius equation shows that with each 10 °C increase in processor temperature, the possibility of malfunction doubles. Additionally, low energy dissipation in processors has additional positive outcomes, such as reduction in cost of cooling, increased systems ergonomics, improved processor lifetime and performance, and even positive effects over electricity bills. Thus, any solution intended for energy savings is intensely valuable for today's processors.

Two methods are popular for solving processor heat-related problems. One method suggests not to change the processors, but to run the processors at desired performance levels by just cooling them better. This solution increases both the physical size of the processor's cooling system and the processor cost. Such thinking offers no definite advantage over cost of power, since the processor dissipates the same amount of power. The second method aims to minimize the disadvantages caused by the cooling system and reduce the power consumption, system ergonomics, system reliability, and the system's lifetime by lowering the energy dissipation of processors. This second method has two alternative approaches.

The first approach, known as Dynamic Voltage/Frequency Scaling (DVFS), decreases energy dissipation by reducing the voltage parameter and hence power dissipation, as per the following equation relating the amount of energy held by a system exhibiting a characteristic capacitance C at rms voltage V:

$$E = \alpha C V^2 \tag{1}$$

where α is the activity factor. Besides being simple, this approach is also very effective for decreasing the energy dissipation of the system, and, thus, it is widely used. The disadvantage of this approach is the transistors are switched more slowly with the low voltage values, and because of this frequency f should also be reduced. Considering the CPU iron-law,

$$T_{exec} = (N \times CPI)/f, \tag{2}$$

for calculating the execution time of an application, in which the execution time is directly proportional to the number of instructions N and cycles per instruction (CPI), and inversely proportional with the processor's frequency f. As a result, the first approach guarantees approximately three times power savings with a linear decrease in processor's performance:

$$P = \alpha C V^2 f. \tag{3}$$

The second approach aims at energy savings by decreasing the capacitance C and the constant, activity factor α. This method is shortly named as the *architectural solution*. In this method activity factor is directly related with capacitors' switching count, capacitance is directly related with capacitors' sizes and wire length. These two alternative methods are orthogonal, and therefore, can be utilized at the same time to achieve greater energy savings. The study in this paper is in the architectural solution category, and specifically targets simultaneous multi-threaded processors.

Simultaneous Multi-Threaded (SMT) processors are very popular due to their improved throughput when running multiple threads and their simplistic design which requires minor modifications to the existing superscalar datapaths. Today, we find examples of SMT processors on server machines such as Intel Xeon as well as inexpensive netbook computers such as the Intel Atom. These SMT processors dissipate considerable amount of energy, which recently became a major issue to be solved to improve the lifetime, reliability, performance and even the die size of these processors.

In SMT processors, there are many datapath resources which are shared (i.e. issue queue, physical register files and caches), and replicated (i.e. reorder buffer, load/store queue and architectural register files). The shared

resources are designed to be larger in size to accommodate the entries coming from multiple threads, whereas replicated resources introduce additional challenges in routing related entries increasing complexity within the processor. As a result, complexity reduction is beneficial for both improving the performance and reducing the energy dissipation on these processors.

In this study, we propose adaptive resizing of issue queue, physical register files, reorder buffer and load/store queue in SMT processors for energy reduction. The proposed method physically partitions a given datapath resource into multiple pieces, and turns off unused (or underutilized) partitions of that resource to reduce the energy dissipation on the processor. The main idea is to achieve maximum energy savings without impacting the performance of the processor. The idea of adaptive resizing of datapath resources is not new. It has been studied for single-threaded superscalar processors before [2–4]. Main contributions of this study are as follows.

- This paper studies the adaptation of the existing algorithms to the SMT processors. To the best of our knowledge, this is one of the first studies that focus on the adaptive resizing of datapath resources in such processors.

- SMT processors are modified superscalar processors that accept instructions from multiple threads into the same instruction pipeline. As a result, some of the datapath resources are shared among multiple threads, and these threads compete for these resources. In this study, we show that a simple control mechanism, which is based on "resource occupancy" and "thread stall"-based statistics is sufficient to save considerable energy on such structures with slight performance gains. These performance results are quite unexpected, especially when they are compared with the findings of superscalar processor study in which similar savings are achieved with an average performance penalty of about 5%.

- The superscalar study in [2–4] targets Pentium-III like processors that combines physical register files and reorder buffers in a single structure. This study, on the other hand, focuses on more contemporary processors used in SMT machines with separate physical register files and reorder buffers. In multithreaded workloads the reorder buffer and physical register requirements may not be correlated at all times, and separate control mechanisms for these resources are proposed in this study.

Remainder of this paper is organized as follows. Section 2 gives detailed background information about energy-efficiency targeting studies. Section 3 describes our methodology. In Section 4, we describe our dynamic resizing algorithm for both queue-type and buffer-type structures. In Section 5, simulation results are discussed. Finally, Section 6 concludes our paper.

1.1. Related work

One of the first works of DVFS is done by Marculescu who suggests using cache misses as a clue for voltage scaling [5]. In this work when a cache miss is discovered it is expressed that processor's activity is separated into two phases: a) independent phase and b) dependent phase. First phase is independent from the instructions which causes cache misses, whereas the second phase depends on these instructions. As a result, while the cache is updated from main memory instructions, the dependent phase waits for the process to complete. At the same time, instructions in the independent phase may continue to execute. However, executing independent instructions with the maximum system voltage supply causes the independent instructions to be finished much earlier than the cache update and causes system to use additional, unnecessary energy consumption. Hence,

Marculescu suggests slowing down the processor (decrease Voltage / Frequency) right after cache misses. Consequently it may be provided that execution of the independent instructions and cache updates will be finalized concurrently.

Weissel and Bellosa [6] offer another approach that adapts the fulfillment frequencies of different hardware incidents like cache misses to running threads. The idea is when OS scheduler makes a context switch, increase or decrease the frequency according to the previously evaluated incident fulfillment frequencies.

Kondo and Nakamura [7] proposes a method which sums up the cache misses periodically and at the end of the periods updates one of the three counters according to the results (being zero, one or more than one). A cost function grades the memory dependence of the executing code upon the weighted summation of these three counters. The voltage and frequency is decreased if the memory dependence of the code is bigger than an upper threshold or increased if the memory dependence of the code is smaller than a lower threshold.

Poellabauer and his crew divide the execution time of an application into two: a) time for calculations and b) memory access time [8]. To estimate the memory dependency of an application, they introduce a new metric that they called memory access rate. This metric evaluates the average cache miss count for each executed instruction. A previously prepared table keeps the resizing factors for each memory access rate, and during the execution time processor is slowed down or speeded up.

Architectural solutions for conserving energy in processors can be categorized into three groups: a) decreasing the activity factor by decreasing the access count to the datapath structures; b) focusing on partitioning the datapath resources into smaller partitions to decrease capacitance factor; and c) decreasing both capacitance factor and the activity factor. The main target of studies of each approach is to gain more than three times power savings (as opposed to linear performance loss). Since an ordinary DVFS algorithm can naturally support this constraint, any work which does not support this constraint becomes totally useless when compared to a standard DVFS algorithm.

Sherwood and Calder have reported datapath resource usage ratios and evaluation using SPEC 95 [9], involving the dynamic behaviors of applications using the popular Simplescalar simulator [10, 11]. In this project three separate structures (i.e., issue queue, physical register files, and reorder buffer) are united into one structure called the register update unit (RUU). The relation through IPC, RUU occupancy ratio, cache misses, branch prediction, value prediction, and address prediction is studied.

Dynamic resizing of the issue queue structure can be considered as one of the first studies similar to our study. In the study by Buyuktosunoglu et al., the issue queue is divided into independent partitions, and the authors aimed for energy savings by turning off the unused partitions according to the instruction level parallelism (ILP) degree [12, 13]. Activity in the issue queue is determined according to the number of ready instructions and it is claimed that this number is directly related to the application's requests. As explained in [2–4], the requests of the application is more related to the number of instructions kept in the structures other than the number of instructions ready to execute on these structures.

In the same year, Folegnani and Gonzalez designed the issue queue as FIFO type queue that allows the out of order execution [14]. In this study, issue queue is partitioned into regions. The queue is of FIFO type, and the region that the instructions inserted is called *the youngest part*. Number of instructions executed and terminated in that part is used as a determination factor for total size of the issue queue needed for the number of partitions. It is suggested to turn on one of the closed regions not influenced by the performance loss.

The first study investigating simultaneous resizing of multiple datapath resources was done by Ponomarev et al. [2–4]. In these studies, dynamic resizing of the partitioned structures by utilization of the average

occupancy ratios of the datapath resources is suggested. Average power savings of 42% in issue queue, 74% in ROB including the PRF (physical register file), and 41% in LSQ (load/store queue) is achieved in a 4-way single-threaded superscalar processor. Average performance loss to provide this savings was limited to 5%.

The work proposed above is supported by Dropsho and his team, using a limited histogram in place of using average occupancy values of resources [15]. However, the system becomes complicated with the histogram mechanism used in that study. There is a processor's configuration difference when two methods are compared. Dropsho et al. preferred a processor with separate ROB and PRF structures, whereas Ponomarev et al. studied combined ROB and PRF structures. Dropsho et al. also resized the cache structures.

Some of the initial works that aim to optimize datapath resources and study several fetch policies for optimizing the throughput by evaluating the requests of the threads executing on simultaneous multi-threaded (SMT) processors are [16–20]. Among them, one work is called Dynamically Controlled Resource Allocation (DCRA), in which threads are classified according to their resource requests and resources are tried to be shared evenly through threads [20]. The methodology aims to increase memory-level parallelism by allocating more resources to the memory bounded threads and allowing multiple cache misses to overlap other than to stall or to remove instructions from the processor.

Robatmili investigated the effect of the several methods, from which suggesting resource partitioning on the processor's performance [21]. The oldest-first based issue queue is separated into multiple partitions for each thread. A similar work is done for single-threaded superscalar processors using ROB [22, 23]. Raasch and Reinhardt studied the structure of ROB in SMT processors, to see whether a bigger structure shared through threads or smaller replicated structures has advantages [24]. As a result, it is found that having partitioned ROB has better performance.

One of the closest studies to this work is done by Sharkey and Ponomarev [25]. By focusing only on ROB, Sharkey suggested dynamic resizing of smaller homogeneous logical partitions of replicated ROBs for each thread. But unlike our study, in that project performance improvement is aimed and no evaluation is done related to energy savings.

2. Dynamic resizing mechanism

In this section, the details of our dynamic resizing mechanism are given. The mechanism is distributed over datapath resources rather than being implemented as a centralized controller. For each of the studied datapath resources, a dedicated finite state machine is utilized.

ROB and LSQ structures in SMT processors are usually replicated (and not shared) for each thread. Both of these structures are in circular FIFO queue type. ROB structure keeps all in-flight instructions in the processor in program order. LSQ structure does the same thing for only memory instructions. Both structures have two pointers indicating the head (H) and the tail (T). Tail pointer indicates the next free entry that the new instruction can be inserted to the structure and head pointer indicates the oldest instruction that can retire or commit.

We divide all targeted datapath resources into equal partitions (Figure 1) to dynamically resize. Thereby energy savings is aimed by bypassing the unused resource partitions. It should be noticed that in Figure 1, the system allows any partition to be turned off. Applying this property to a FIFO queue structure might cause logical problems. Consider partition 1 in Figure 1 is turned off, and queue structure starts from partition 0 and continues to partition 2. In that case, the reactivation of partition 1 for reuse, will damage the integrity of the

queue structure. To prevent such problems, we only allow the activation and turning off of the last available partition for all structures we studied.

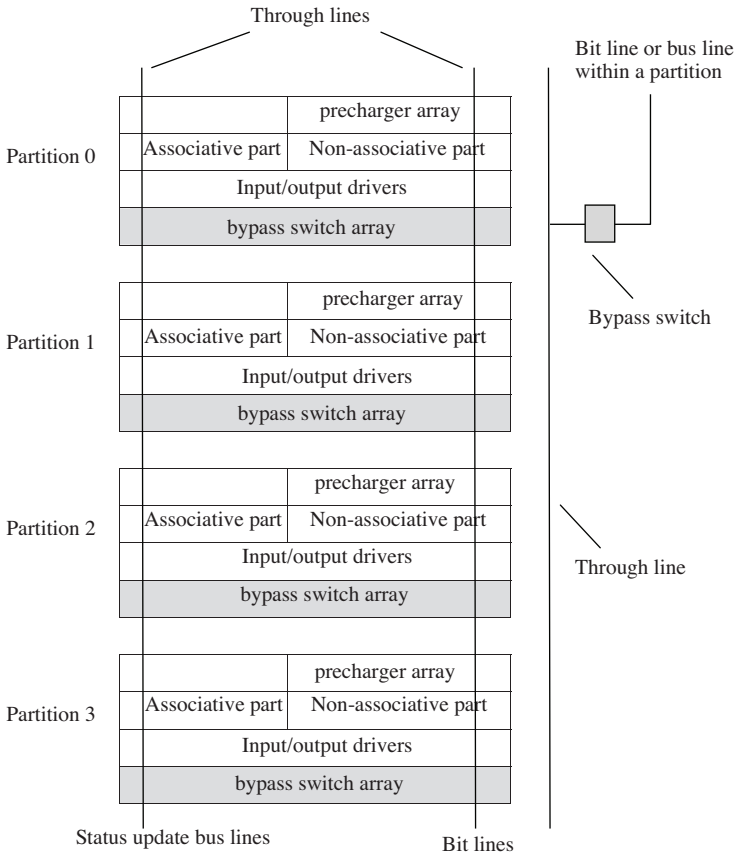

Figure 1. Partitioned resource structure.

The finite state machine for ROB and LSQ structures is shown in Figure 2. Starting from the stable phase, the algorithm attempts upsize or downsize phases. Unfortunately, these decisions cannot be satisfied immediately for circular FIFO queue structures due to possible integrity problems. It is quite possible that these phases cannot be serviced in a single clock cycle, and, meanwhile, new decisions might be taken. For instance, during a Resource Downsize Phase, a Resource Upsize Decision can be taken, or vice versa. In those cases, we give priority to Resource Upsize Decision because of our performance concerns.

Furthermore, in our current control mechanism, we aim to activate or deactivate one partition at a time. In an alternative method, more than one partition (or maybe all) can be activated or deactivated. However, we found that these aggressive techniques can create oscillations in the control mechanism and harm the system stability and performance. A more thorough and detailed study is needed to investigate their integration to the current control mechanism.

The instant positions of the Head (H) and Tail (T) pointers directly affect the time required to accomplish the resource resizing process. In the scenario given in Figure 3(a), the resource upsize or downsize is not immediately possible. If a resource downsize decision is taken, the position of the head should be lower than the position of the tail. In Figure 3(b), this is the case, and both the activation and deactivation of the last partition is possible.

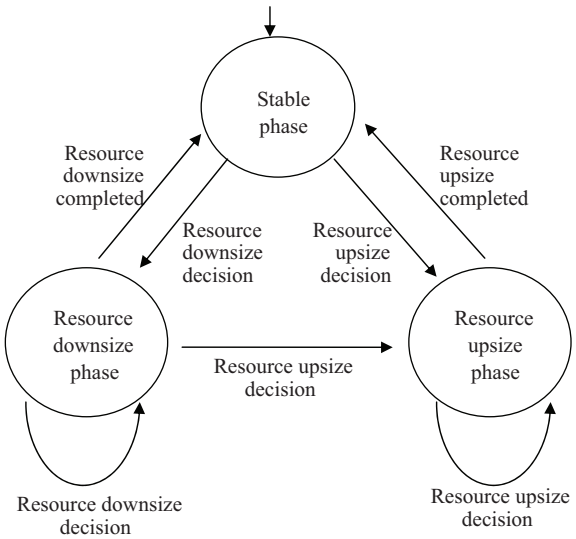

Figure 2. Resizing algorithm for queue structures.

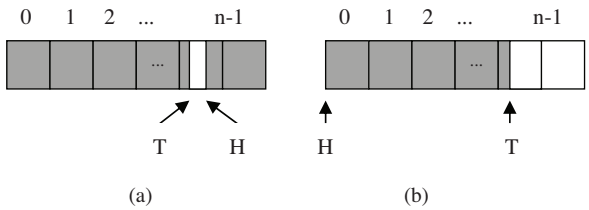

Figure 3. Two scenarios for resource resizing in queue-type structures.

The Resource Downsize Decision can be taken at the end of predefined *Sampling Periods*. In our tests, the sampling period is chosen to be 32 Kcycles. Before taking the decision, average resource occupancies within the preceding period are collected and then these numbers are compared with the occupancies inside the currently active partitions. For instance, let us consider the datapath resource depicted in Figure 1, and assume that the first three partitions are active and the last one is gated off at the end of a sampling period. When it is found that the average resource occupancy can fit in only the first two partitions at that time, then the current last partition (partition 2) is decided to be turned off (i.e. a Resource Downsize Decision is taken). Otherwise, the Resource Downsize Decision is not taken, and the resizing mechanism continues with its existing configuration.

For the Resource Upsize Decision, our past experiences show that waiting until the end of a specific period may degrade the system performance. Therefore, we decided to apply the same upsizing technique that we use for single-threaded superscalar processors. In that case, the Resource Upsize Decision can be taken at any time. To be able to trigger an upsize decision, a new counter is added to count the number of stalls for each datapath resource. This counter is checked at every cycle, and when it exceeds a predefined threshold value, the Upsize Threshold, the Resource Upsize Decision is taken. After the decision, the corresponding stall counter is immediately reset. For instance, again, let us consider the datapath resource depicted in Figure 1, and assume that the first three partitions are active and the last one is turned off at some point during execution. When a new allocation request arrives and all active partitions are full, the new item cannot be immediately inserted, and, as a result, a stall occurs. After each resource stall, the corresponding stall counter is incremented. When,

the counter exceeds its threshold value, the last partition (partition 3) is decided to be activated (i.e. a Resource Upsize Decision is taken).

IQ and PRF datapath resources are shared among all the threads in SMT processors. Moreover, they contain instructions and data in an out-of-program-order fashion and, therefore, are both implemented in buffer structures. Resizing of these resources has fewer complications compared to that of ROB and LSQ. For instance, a resource upsize can be performed right after a Resource Upsize Decision without any need for a specific Resource Upsize Phase. After a resource downsize decision, on the other hand, the mechanism still requires a similar (but somewhat less complicated) Resource Downsize Phase for waiting the last partition to be completely unoccupied. In Figure 4, the finite state machine for these buffer structures is depicted.

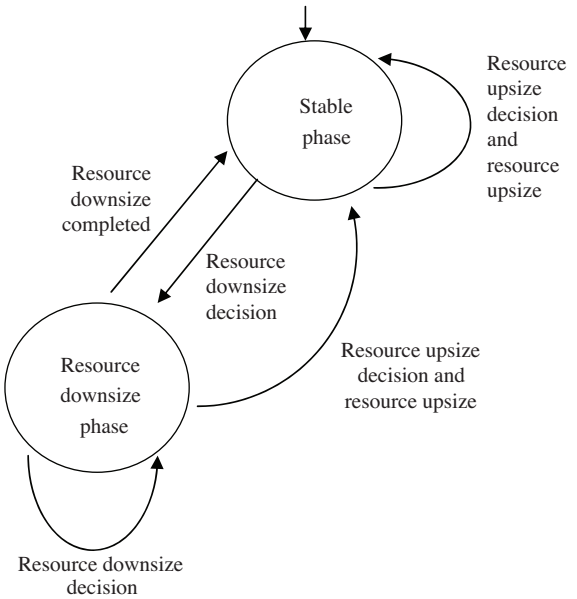

Figure 4. Resizing algorithm for buffer structures.

When an upsize or downsize decision is made, and the corresponding phase is over, the partition cannot be immediately turned on or off, since this type of operation can easily create $\mathrm{d}i/\mathrm{d}t$ spikes on the processor. In our simulation, we accounted the possibility of this problem, and allowed 300 cycles for completely turning on and off a partition.

3. Methodology

In this study several datapath statistics, such as resource occupancies, cache miss rates, branch misprediction rates, applications types (i.e. memory- or CPU-intensive), fairness of resource sharing through threads, and other similar metrics are collected and examined separately or together to investigate the effects of dynamic resizing mechanism. For the SMT simulation we used the M-Sim simulator [26]. This simulator also provides an integrated power estimator tool called Wattch [27].

In Table 1, the simulation parameters are given, in detail. Note that the *Sampling Period* value is chosen as 32 Kcycles, and the *Upsize Threshold* value is chosen as 32 Kstalls (i.e. there should be 32 K stalls to each resource, until a resource upsizing decision is triggered.) These are empirically determined values, which gave

near optimal results in our tests. We ran all simulations in SimPoint [28] ranges. We arranged 12 mixtures, application couples, as a result of our literature survey [18–21]. Mixtures are formed of integer (INT) and floating point (FP) applications, and they are also selected according to their characteristics, such as their instruction level parallelism (ILP) degree, memory-boundedness (MEM), and a mixture of both ILP and memory bounded (MIXT). The details of these application mixtures are given in Table 2.

Table 1. Processor configuration used throughout the tests.

Parameter	Configuration
Machine width	8-wide fetch, 8-wide issue, 8-wide commit
Window size	64-entry IQ, 96-entry ROB, 48-entry LSQ, 192-entry PRF
L1 I-Cache	512KB, 32- cache block size, 2-way set associative, least recently used (LRU) block replacement strategy
L1 D- Cache	512KB, 32-cache block size, 4-way set associative, LRU block replacement strategy
L2 Cache	1024KB, 128- cache block size, 8-way set associative, LRU block replacement strategy
TLB	(I) 16-entry, 4096-page size, 4-way set associative, LRU block replacement strategy, (D) 32-entry, 4096- page size, 4-way set associative, LRU block replacement strategy
Functional Units	8-INT ALU, 3-INT MULT, 8-FP ALU, 3-FP MULT
Sampling Period	32 Kcycles
Upsize Threshold	32 Kstalls

Table 2. Application mixtures.

Mix #	Appl.#1	Appl.#2	Characteristic	Type #1	Type #2
Mix1	Twolf	Art	MEM	INT	FP
Mix2	Equake	Parser	MEM	FP	INT
Mix3	Mgrid	Mcf	MEM	FP	INT
Mix4	Gzip	Bzip2	ILP	INT	INT
Mix5	Gcc	Mesa	ILP	INT	FP
Mix6	Vortex	Wupwise	ILP	INT	FP
Mix7	Twolf	Gzip	MIXT	INT	INT
Mix8	Art	Bzip2	MIXT	FP	INT
Mix9	Equake	Gcc	MIXT	FP	INT
Mix10	Parser	Mesa	MIXT	INT	FP
Mix11	Mgrid	Vortex	MIXT	FP	INT
Mix12	Mcf	Wupwise	MIXT	INT	FP

4. Tests and results

In this section, we discuss the results of the proposed resizing mechanism, in detail. First, in Figures 5 and 6, we show the simulation results of two application mixtures (Mix1 and Mix3) in a short execution window, respectively. Mix1 (see Figure 5) is a nice example for showing that sudden changes in benchmark behavior are possible. Here, resource usage increases and decreases, instantly. Applications with such behavior are tricky to handle for any control mechanism causing it to be unstable, oscillate, and, as a result, degrade processor performance. However, for our chosen sampling period and the upsize threshold, we successfully capture the

behavior of all mixtures, as the dynamic sizes of the resources (the right-hand side graphs) successfully track down the changes in the resource occupancies (the left-hand side graphs).

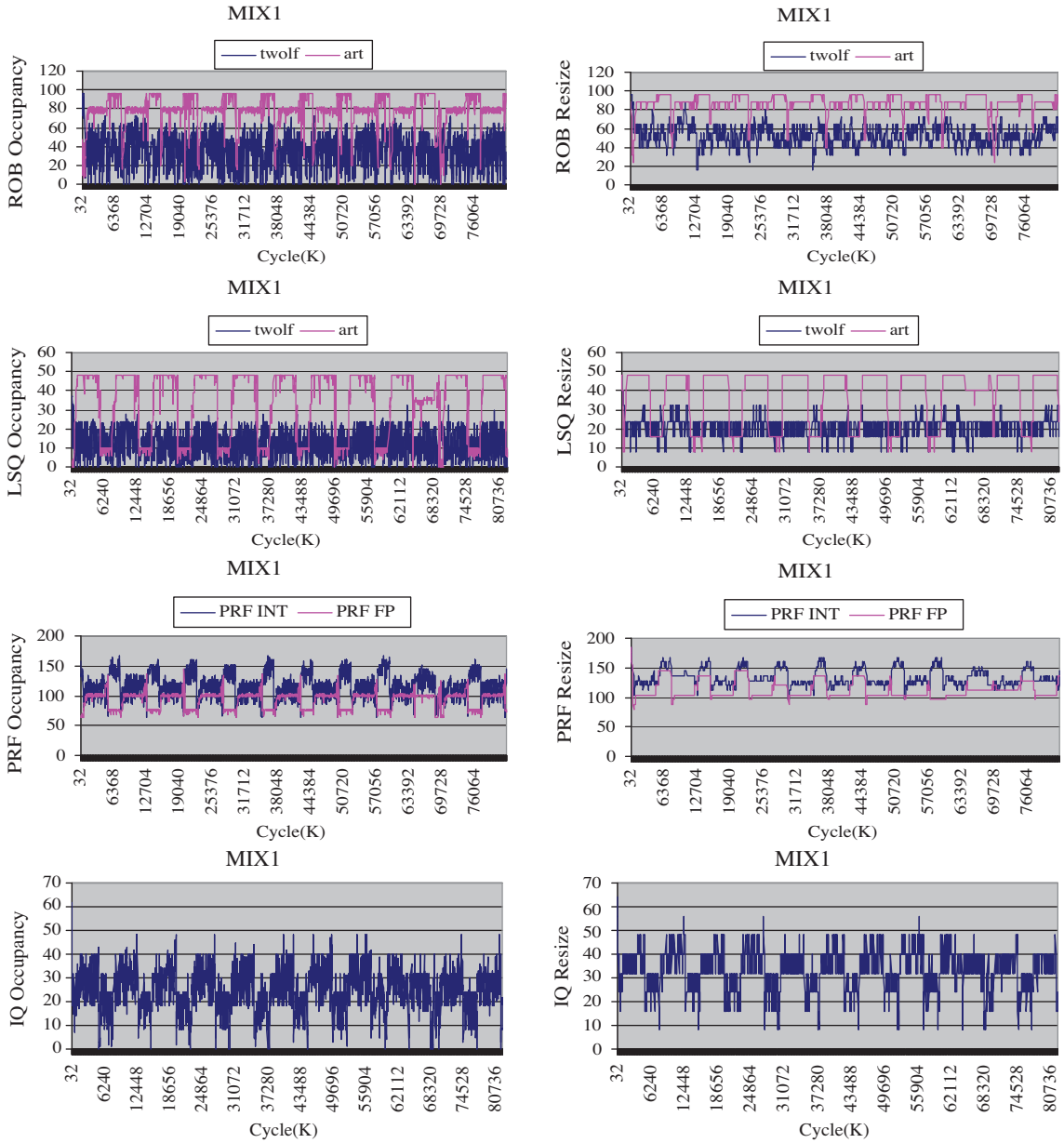

Figure 5. MIX 1 (twolf – art) results.

In Figure 6, IQ graphs show an interesting behavior in Mix3. In spite of IQ is a shared resource, for some periods its usage is at minimum. As a result, we find that even shared resources give promising results for resource downsizing in SMT processors. Similar savings are also observed in other shared resources, i.e. physical register files, as well.

The performance results and fraction of active occupancy of resources are given in Table 3 and Table 4, respectively. These results give a general idea of performance and energy savings that can be achieved by the

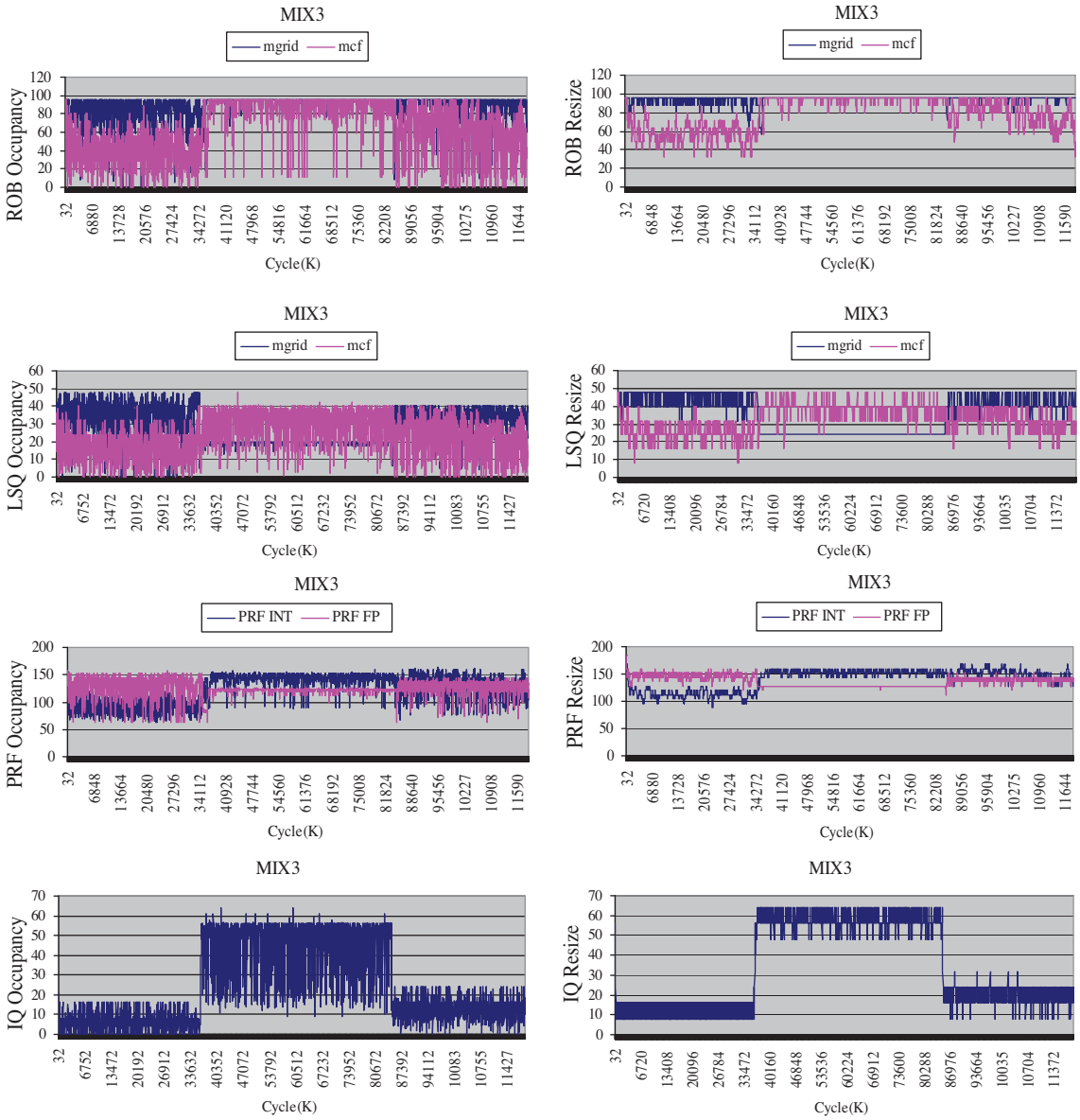

Figure 6. MIX 3 (mgrid – mcf) results.

utilization of our proposed mechanism. From Table 4, it can be noticed that average occupancy percentages, which are compared to the baseline processor, may differ for each application and each resource. Instructions per cycle (IPC) and total instructions per cycle (TPC) drop percentages can be both negative (performance gain) and positive (performance drop).

At first, performance improvements might be found as an interesting outcome, especially when the resources are downsized. However, this is quite usual for speculative, out-of-order superscalar processors. When resource sizes are decreased, speculative instructions are prevented to be inserted into resources. This way removing misspeculated instructions from resources can be done quicker since the resource sizes are smaller. When the results are evaluated we also found that aggressive upsizing mechanism, for performance concerns, keeps resource sizes higher.

Table 3. Performance comparison of baseline processor and the one that integrates the proposed configuration.

Mix	Benchmark	IPC Baseline	IPC prop.conf.	IPC Drop%	TPC Baseline	TPC prop.conf.	TPC Drop%
Mix1	Twolf	0.38	0.39	-2.63	0.67	0.66	1.49
	Art	0.29	0.27	6.90			
Mix2	Equake	0.37	0.39	-5.41	1.16	1.14	1.72
	Parser	0.79	0.75	5.06			
Mix3	Mgrid	0.26	0.29	-11.54	0.29	0.32	-10.34
	Mcf	0.03	0.03	0.00			
Mix4	Gzip	1.50	1.74	-16.00	2.74	2.34	14.60
	Bzip2	1.24	0.60	51.61			
Mix5	Gcc	1.29	1.21	6.20	2.81	2.61	7.12
	Mesa	1.52	1.40	7.89			
Mix6	Vortex	1.38	1.29	6.52	2.76	2.44	11.59
	Wupwise	1.38	1.15	16.67			
Mix7	Twolf	0.50	0.50	0.00	1.70	1.79	-5.29
	Gzip	1.20	1.29	-7.50			
Mix8	Art	0.34	0.31	8.82	1.56	1.48	5.13
	Bzip2	1.22	1.17	4.10			
Mix9	Equake	0.41	0.41	0.00	1.58	1.48	6.33
	Gcc	1.17	1.07	8.55			
Mix10	Parser	0.82	0.82	0.00	2.08	2.03	2.40
	Mesa	1.26	1.21	3.97			
Mix11	Mgrid	0.87	0.83	4.60	2.41	2.19	9.13
	Vortex	1.54	1.36	11.69			
Mix12	Mcf	0.03	0.03	0.00	0.12	0.22	-83.33
	Wupwise	0.09	0.19	-111.1			

Here, we focus on one of the mixtures, and discuss the results, in detail. However, these findings are similar in all the mixtures that we studied. When we check the results of Mix3, from both Table 3 and Table 4, we see that *mcf* has more LSQ occupancy than *mgrid*, on average. We may conclude that *mcf* executes more memory-bounded instructions than *mgrid*. Thus, *mcf* waits for stalls which decrease its IPC value and throughput. We also see that the ROB occupancy of *mcf* is quite high (98.96%). Since any instruction inserted into the ROB cannot be removed before its execution finished. That causes *mcf*'s ROB to get full in a short period of time. Memory instructions, which stay at the head of ROB but cannot commit due to cache misses, affects the IPC value of *mcf*. *Mgrid*, on the other hand, executes more instructions per cycle, and it retires more instructions from its ROB and other related resources.

When we focus on the IQ, which is not a replicated resource, we see that *mcf* fills the IQ with its instructions, and causes 98.44% occupancy. However, we see that we still have some energy savings on this resource. Finally, when we check the integer physical register file (INT-PRF) and the floating-point physical register file (FP-PRF) structures, we see that occupancy percentage of the INT-PRF is higher than occupancy percentage of the FP-PRF. Table 2 clarifies the reason behind this observation: the *mcf* is an INT-type application, whereas *mgrid* is a FP-type one. Since *mcf* cannot commit its instructions as fast as *mgrid*, INT-PRF occupancy percentage stays high. Since TPC does not decrease, we may gain energy savings without any performance loss.

Table 4. Average occupancy percentages of resized structures.

Mix	Benchmark	Average occupancy, %				
		ROB	LSQ	IQ	INT-PRF	FP-PRF
Mix 1	Twolf	72.92	52.08	70.31	82.29	60.94
	Art	77.08	54.17			
Mix 2	Equake	86.46	83.33	85.94	74.48	68.75
	Parser	59.38	39.58			
Mix 3	Mgrid	18.75	16.67	98.44	79.17	42.71
	Mcf	98.96	56.25			
Mix 4	Gzip	34.38	20.83	32.81	61.98	33.33
	Bzip2	23.96	16.67			
Mix 5	Gcc	35.42	31.25	25.00	60.42	43.75
	Mesa	37.50	33.33			
Mix 6	Vortex	44.79	35.42	20.31	63.02	51.56
	Wupwise	44.79	29.17			
Mix 7	Twolf	68.75	50.00	60.94	76.56	39.06
	Gzip	28.13	18.75			
Mix 8	Art	81.25	66.67	56.25	77.60	58.85
	Bzip2	63.54	45.83			
Mix 9	Equake	92.71	87.50	78.13	59.38	78.13
	Gcc	31.25	29.17			
Mix 10	Parser	65.63	39.58	50.00	70.31	40.63
	Mesa	30.21	25.00			
Mix 11	Mgrid	76.04	62.50	12.50	55.21	71.88
	Vortex	41.67	33.33			
Mix 12	Mcf	98.96	56.25	98.44	83.33	40.10
	Wupwise	19.79	10.42			

Table 5. Power results of all mixtures.

	Average Total Power per Instruction		
	Baseline	Proposed configuration	Power Reduction, %
mix1	345.86	310.33	10.27
mix2	63.27	53.79	14.98
mix3	114.76	104.55	8.90
mix4	75.37	65.49	13.11
mix5	62.08	52.40	15.58
mix6	63.52	55.94	11.93
mix7	47.99	42.05	12.38
mix8	315.14	285.20	9.50
mix9	64.73	54.47	15.86
mix10	60.00	51.58	14.03
mix11	118.69	105.50	11.11
mix12	48.93	45.16	7.70

Finally, Table 5 shows the average values of total power per instruction for the baseline and the proposed configuration in all mixtures. Here, we see that, by the application of our proposed mechanism on SMT processors, we get consistent power savings for all simulated mixtures, and the total processor power reduction is more than 12%, on the average.

5. Conclusion

In this study, we propose the adaptation of an architectural technique that was originally proposed for superscalar processors, to SMT processors, for reducing their energy dissipation. The aim is to conserve energy by physically partitioning replicated datapath resources such as reorder buffers (ROB), load/store queues (LSQ) as well as shared resources such as the issue queue (IQ) and physical register files (PRF), and then dynamically turning off unused partitions on each of these resources. The dynamic resizing algorithm turns off 45% of the ROB, 59% of the LSQ, 43% of the IQ, 30% of the integer PRF and, finally, 48% of the floating PRF, on the average. These numbers are directly related to the average energy savings we can achieve by utilizing our proposed method. As a result, our method reduces total processor power by more than 12% on the average, while improving processor performance by 3.2%, on the average across all simulated application mixtures. These energy savings are aligned with our previous findings for superscalar processors. Performance-wise, the mechanism proposed in this study gives better results compared to our previous study.

References

[1] Pollack, F., "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies," Micro32 conference key note, 1999.

[2] Ponomarev, D., Kucuk, G., Ghose, K., "Reducing Power Requirements of Instruction Scheduling through Dynamic Allocation of Multiple Datapath Resources," *in Proceedings of the 34th International Symposium on Microarchitecture*, 2001.

[3] Ponomarev, D., Kucuk, G., Ghose, K., "Dynamic Allocation of Datapath Resources for Low Power," *in Proceedings of the Workshop on Complexity-Effective Design,* held in conjunction with ISCA-28, 2001.

[4] Ponomarev, D., Kucuk, G., Ghose, K., "Dynamic Resizing of Superscalar Datapath Components for Energy Efficiency," *in IEEE Transactions on Computers*, vol. 55, No. 2, Feb 2006, pp.192-213.

[5] Marculescu, D., "On the Use of Microarchitecture-Driven Dynamic Voltage Scaling," *in Proceedings of the Workshop on Complexity-Effective Design*, 2000.

[6] Weissel, A., Bellosa, F., "Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management," *in Proceedings of the International Conference on Compilers, Architecture, ad Synthesis for Embedded Systems*, 2002.

[7] Kondo, M., Nakamura, H., "Dynamic Processor Throttling for Power Efficient Computations," *in Proceedings of the Workshop on Power-Aware Computer Systems*, 2004.

[8] Poellabauer, C., Singleton, L., Schwan, K., "Feedback Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications," *in IEEE Real Time and Embedded Technology and Applications Symposium*, 2005.

[9] Standard Performance Evaluation Corporation, http://www.spec.org

[10] Simplescalar LLC, http://www.simplescalar.com

[11] Sherwood, T., Calder, B., "Time Varying Behavior of Programs," Technical Report No. CS99-630, Dept. of Computer Science and Engineering, University of California San Diego, 1999.

[12] Buyuktosunoglu, A., Schuster, S., Brooks, D., Bose, P., Cook, P., Albonesi, D., "An Adaptive Issue Queue for Reduced Power at High Performance," *in Proceedings of the Workshop Power-Aware Computer Systems*, 2000.

[13] Buyuktosunoglu, A., Albonesi, D., Schuster, S., Brooks, D., Bose, P., Cook, P., "A Circuit Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors," *in Proceedings of the Great Lakes Symposium VLSI Design*, 2001.

[14] Folegnani, D., Gonzalez, A., "Energy-Effective Issue Logic," *in Proceedings of the International Symposium on Computer Architecture*, 2001.

[15] Dropsho, S. et al., "Integrating Adaptive On-Chip Structures for Reduced Dynamic Power," *in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[16] Tullsen, D. et al., "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *in Proceedings of the International Symposium on Computer Architecture*, 1996.

[17] Tullsen, D. et al., "Handling Long-Latency Loads in a Simultaneous Multi-Threaded Processor," *in Proceedings of the International Symposium on Microarchitecture*, 2001.

[18] Cazorla, F. et al., "Improving Memory Latency Aware Fetch Policies for SMT Processors," *in Proceedings of the International Symposium on High Performance Computing*, 2003.

[19] El-Moursy, A., Albonesi, D., "Front-End Policies for Improved Issue Efficiency in SMT Processors," *in Proceedings of the International Symposium on High-Performance Computer Architecture*, 2003.

[20] Cazorla, F. et al., "Dynamically Controlled Resource Allocation in SMT Processors," *in Proceedings of the International Symposium on Microarchitecture*, 2004.

[21] Robatmili, B. et al., "Thread-Sensitive Instruction Issue for SMT Processors," *in Computer Architecture News*, 2004.

[22] Kucuk, G., Ergin, O., Ponomarev, D., Ghose, K., "Distributed Reorder Buffer Schemes for Low Power," *in Proceedings of the International Conference on Computer Design*, 2003.

[23] Kucuk, G. Ponomarev, D., Ergin, O., Ghose, K., "Complexity-Effective Reorder Buffer Designs for Superscalar Processors," *in IEEE Transactions on Computers,* vol. 53, No. 6, 2004, pp.653-665.

[24] Raasch, S., Reinhardt, S., "The Impact of Resource Partitioning on SMT Processors," *in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2003.

[25] Sharkey, J., Balkan, D., Ponomarev D., "Adaptive Reorder Buffers for SMT Processors," *in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2006.

[26] Sharkey, J., Ponomarev, D., Ghose, K., "M-Sim: A Flexible, Multithreaded Architectural Simulation Environment," Technical Report CS-TR-05-DP01, Department of Computer Science, State University of New York at Binghamton, October 2005.

[27] Brooks, D., Tiwari, V., Martonosi, M., "Wattch: A Framework for Architecture-Level Power Analysis and Optimizations," *in Proceedings of the International Symposium on Computer Architecture*, pp.83-94, 2000.

[28] Hamerly, G., Perelman, E., Calder, B., "How to Use SimPoint to Pick Simulation Points," ACM SIGMETRICS Performance Evaluation Review, 2004.