Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources*

Dmitry Ponomarev, Gurhan Kucuk, Kanad Ghose
Department of Computer Science
State University of New York, Binghamton, NY 13902–6000
e–mail:{dima, gurhan, ghose}@cs.binghamton.edu

Abstract

The "one–size–fits–all" philosophy used for permanently allocating datapath resources in today's superscalar CPUs to maximize performance across a wide range of applications results in the overcommitment of resources in general. To reduce power dissipation in the datapath, the resource allocations can be dynamically adjusted based on the demands of applications. We propose a mechanism to dynamically, simultaneously and independently adjust the sizes of the issue queue (IQ), the reorder buffer (ROB) and the load/store queue (LSQ) based on the periodic sampling of their occupancies to achieve significant power savings with minimal impact on performance. Resource upsizing is done more aggressively (compared to downsizing) using the relative rate of blocked dispatches to limit the performance penalty. Our results are validated by the execution of SPEC 95 benchmark suite on a substantially modified version of Simplescalar simulator, where the IQ, the ROB, the LSQ and the register files are implemented as separate structures, as is the case with most practical implementations. For the SPEC 95 benchmarks, the use of our technique in a 4–way superscalar processor results in a power savings in excess of 70% within individual components and an average power savings of 53% for the IQ, LSQ and ROB combined for the entire benchmark suite with an average performance penalty of only 5%.

Keywords: superscalar processor, energy–efficient datapath, power reduction, dynamic instruction scheduling.

1. Introduction

Contemporary superscalar datapath designs attempt to push the performance envelope by employing aggressive out–of–order instruction execution mechanisms. These processors are also designed following a "one–size–fits–all" philosophy resulting in the permanent allocation of datapath resources to maximize performance across a wide range of applications. Earlier studies have indicated that the overall performance, as measured by the number of instructions committed per cycle (IPC), varies widely across applications [19]. The IPC also changes quite dramatically within a single application, being a function of the program characteristics (natural instruction–level parallelism – ILP) and that of the datapath and the memory system. As the natural ILP varies in a program, the usage of datapath resources also changes significantly.

It is well–documented that the major power/energy sinks in a modern superscalar datapath are in the dynamic instruction scheduling components (consisting of the issue queue (IQ), the reorder buffer (ROB), the load/store queue (LSQ) and the physical registers) and the on–chip caches. As indicated in [23], as much as 55% of the total power dissipation occurs in the dynamic instruction scheduling logic, while 20% to 35% (and sometimes higher) of the total power is dissipated within the on–chip cache hierarchy. It is therefore not surprising to see a fair amount of recent research being directed towards the reduction of the energy dissipation within these components.

In this paper we introduce a technique for reducing the power dissipation within the IQ, the LSQ and the ROB in a coordinated fashion. The basic approach is a technology independent solution at the microarchitectural level that divides each of the IQ, the LSQ and the ROB into incrementally allocable partitions. Such partitioning effectively permits the active size of each of these resources (as determined by the number of currently active partitions) to be varied dynamically to track the actual demands of the application, and forms the basis of the power savings technique presented here. We also show how simple circuit–level implementation techniques can be naturally augmented into our multi–partitioned resource allocation scheme to achieve substantial power savings without any compromise of the CPU cycle time. Our basic approach for reducing the power dissipation within the IQ, the LSQ and the ROB is orthogonal to the approach taken by the more traditional techniques that use voltage and frequency scaling; such techniques can be deployed in conjunction with our scheme.

The technique proposed here uses sampled estimates of the occupancies of the IQ, the LSQ and the ROB to turn off unused (i.e. unallocated) partitions within these resources to conserve power. As the resource demands of the application go up, deactivated partitions are turned back on to avoid any undue impact on performance. The proposed approach is thus effectively a feedback control system that attempts to closely track the dynamic demands of an application and allocates "just the right amount of resources at the right time" to conserve power.

Recent approaches for power energy reduction based on the broad notion of feedback control, as discussed in Section

* supported in part by DARPA through contract number FC 306020020525 under the PAC–C program, and the NSF through award no. MIP 9504767 & EIA 9911099.

2, are based on the use performance metrics like the IPC, commit rates from newly allocated regions of a resource or sensed temperatures as well as continuous measures of the occupancy of a single resource (namely the IQ). Rather than use IPCs and other measures of performance such as cache hit rates, misprediction rates or physical parameters like sensed temperature to drive dynamic resource allocations and deallocations, we use the immediate history of actual usages of the IQ, the LSQ and the ROB to control their effective sizes independently. The actual resource usages are not monitored continuously but sampled periodically, keeping hardware requirements simple. Resources are downsized, if need be, at the end of periodic update intervals by deallocating one or more partitions and turning them off. Additionally, resource allocations are increased before the end of an update period if the resources are fully utilized and cause instruction dispatching to be blocked for a predetermined number of cycles. This relatively aggressive strategy for increasing resource allocation allows us to severely limit the performance loss and yet achieve significant power savings.

The rest of the paper is organized as follows. Section 2 describes related work and the motivations for the present work. Section 3 describes the superscalar datapath assumed for this study and identifies sources of power dissipation in the major components of the datapath such as the IQ, the ROB and the LSQ. Our simulation methodology is described in Section 4. Section 5 explains the complications involved in resizing of datapath components in general. In Section 6, we study the correlations among the occupancies of the IQ, the ROB, and the LSQ. Our resizing strategy is described in Section 7. In Section 8 we discuss the simulation results, followed by our conclusions in Section 9.

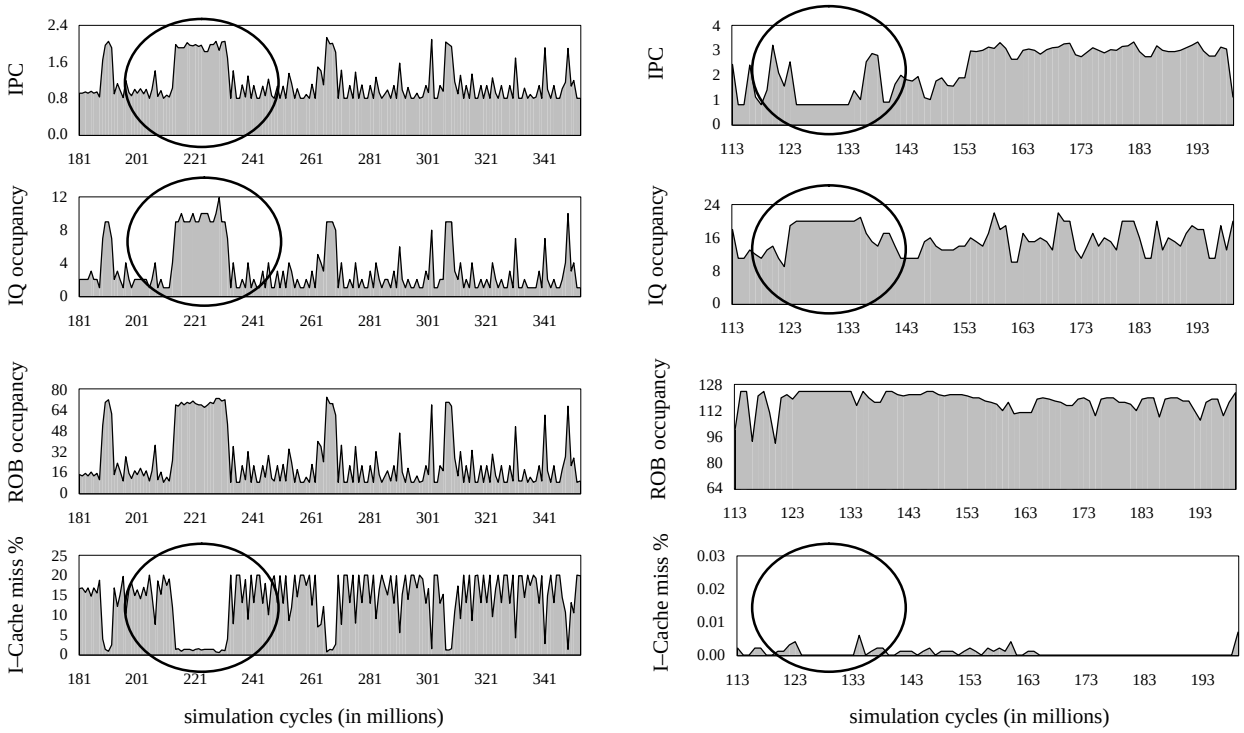
2. Related Work and Motivations

The technique presented in this paper for reducing power dissipation in the instruction scheduling logic hinges on the estimation of resource occupancies. The resource usages and the dynamic behavior of the SPEC 95 benchmarks were reported in [21] using the well–used Simplescalar simulator [8], where the reorder buffer, the physical registers and the issue queue are integrated into an unified structure, called the Register Update Unit (RUU). The correlations among the IPC, RUU occupancy, cache misses, branch prediction, value prediction and address prediction were also documented in [21]. We extend the study of [21] to architectures where the issue queue and the reorder buffer are implemented as distinct resources, as in most practical implementations of superscalar datapaths. We also study the correlations among the usage of these resources. Our studies show why a distributed, dynamic allocation of these resources is needed for reducing the power/energy dissipation (Section 6).

Dynamic resource allocations within a *single* datapath component (the IQ) for conserving power was studied in [9, 10, 12]. Specifically, in [9, 10], the authors explored the design of an adaptive issue queue, where the queue entries were grouped into independent modules. The number of modules allocated was varied dynamically to track the ILP; power savings was achieved by turning off unused modules. In [9, 10], the activity of an issue queue entry was indicated by its ready bit (which indicates that the corresponding instruction is ready for issue to a function unit). The number of active issue queue entries, measured on a cycle–by–cycle basis, was used as a direct indication of the resource demands of the application. We believe that resource occupancies provide a more accurate measure of the resource needs of the program rather than the number of ready entries within the active modules of the IQ. Our simulations show that the number of ready entries is considerably lower than the number of allocated entries in the IQ on the average. Consequently, downsizing the IQ based on the ready bits is very aggressive and the performance drop for some applications may be fairly significant, as can be seen from the results shown in [10]. In addition, the notion of ready entries is only applicable to the IQ. Dynamic allocation within the ROB and the LSQ requires the consideration of alternative strategies.

The performance drop reported in [9, 10] was limited by monitoring the IPC and ramping up the issue queue size if the measured IPC was lower than the IPC obtained by previous measurement by a constant factor. This works under the assumption that the only cause of the IPC drop is the lack of datapath resources. In reality, the IPC drop may be attributed to a variety of other factors, including the higher branch misprediction rate and the higher I–cache miss rate. Figure 1 demonstrates the behavior of two SPEC 95 floating point benchmarks, where the IPC drop is caused by different reasons. Measurements of the IPCs, the average IQ and ROB occupancies and the I–cache miss rates were taken every 1 million cycles. Each of these measurements reflect the benchmark's behavior within the most recent window of 1 milion cycles. Results are shown for the execution of 200 million instructions after skipping the first 200 million for two benchmarks. The configuration of the simulated system used was as given in Section 4. The resource upsizing strategy proposed in [9, 10] is an adequate response to the IPC drop for *the hydro2d* benchmark (Figure 1 (b)), where the drop is caused by the transition from a period of high–ILP execution to a low–ILP one. Here, the datapath resource usage increases due to the data dependencies over the instructions with long latencies and the allocation of new resources is justified. In contrast, the IPC drop during the execution of *fpppp* benchmark is mostly caused by the increased percentage of the I1–cache misses (Figure 1(a)). The resources remain underutilized and ramping up the resource sizes in such situation results in higher power dissipation without any impact on performance. Similarly, when the branch misprediction rate increases, more instructions are flushed from the IQ, the ROB and the LSQ, again resulting in the underutilization of these resources.

To summarize, the IPC drop alone (as used in [9, 10]) is not a sufficient driver for the resizing decision. In the very least, it must be used in combination with the I–cache miss rate and the branch misprediction rate statistics and possibly other metrics. In this paper, we monitor the resource undercommitment by recording the number of cycles when



a) Behavior of **fpppp** benchmark

b) Behavior of **hydro2d** benchmark

Figure 1. Dynamic behavior of two SPEC95 floating point benchmarks

dispatch blocks because of the non–availability of free entries within the IQ, the ROB or the LSQ. This information, directly indicating the increased resource demands, is then used to drive the upsizing decision.

In [12], Folegnani and Gonzalez introduced a FIFO issue queue that permitted out–of–order issue but avoided the compaction of vacated entries within the valid region of the queue to save power. The queue was divided into regions and the number of instructions committed from the most–recently allocated issue queue region in FIFO order (called the "youngest region") was used to determine the number of regions within the circular buffer that was allocated for the actual extent of the issue queue. To avoid a performance hit, the number of regions allocated was incremented by one periodically; in–between, also at periodic intervals, a region was deactivated to save energy/power if the number of commits from the current youngest region was below a threshold. The energy overhead of the control logic for doing this resizing was not made clear. Additional energy savings were documented by not activating forwarding comparators within entries that are ready for issue or entries that are unallocated. In [17], we have introduced an issue queue design that achieves significant energy savings using a variety of techniques, including the use of zero–byte encoding, comparators that dissipate energy on a match and a form of dynamic activation of the accessed regions of the issue queue using bit–line segmentation. Many of the techniques used in [17] can be employed in conjunction with the scheme proposed in this paper to achieve a higher level of power savings.

In reality, the usage of *multiple* resources within a datapath is highly correlated, as seen from the results presented in this paper (and in some of the work cited below). The allocation of multiple datapath resources to conserve power/energy was first studied in the context of a multi–clustered datapath (with non–replicated register files) in [25], where dispatch rates and their relationship to the number of active clusters were well–documented. A similar but a more explicit study was recently reported in [5], for the multi–clustered Compaq 21264 processor with replicated register files. The dispatch rate was varied between 4, 6 and 8 to allow an unused cluster of function units to be shut off completely. The dispatch rate changes were triggered by the crossing of thresholds associated with the floating point and overall IPC, requiring dispatch monitoring on a cycle–by–cycle basis. Fixed threshold values as used in [5] were chosen from the empirical data that was generated experimentally. Significant power savings within the dynamic scheduling components were achieved with a minimum reduction of the IPC. The dynamic allocation of the reorder buffer – a major power sink – was left completely unexplored in this study.

In [15], the dispatch/issue/commit rate ("effective pipeline width") and the number of reorder buffer entries (the unified RUU size of the Simplescalar simulator) was dynamically adjusted to minimize the power/energy characteristics of hot spots in programs. This was done by dynamically profiling the power consumption of each program hot spot on all viable ROB size and pipeline width combinations. The configuration that resulted in the lowest

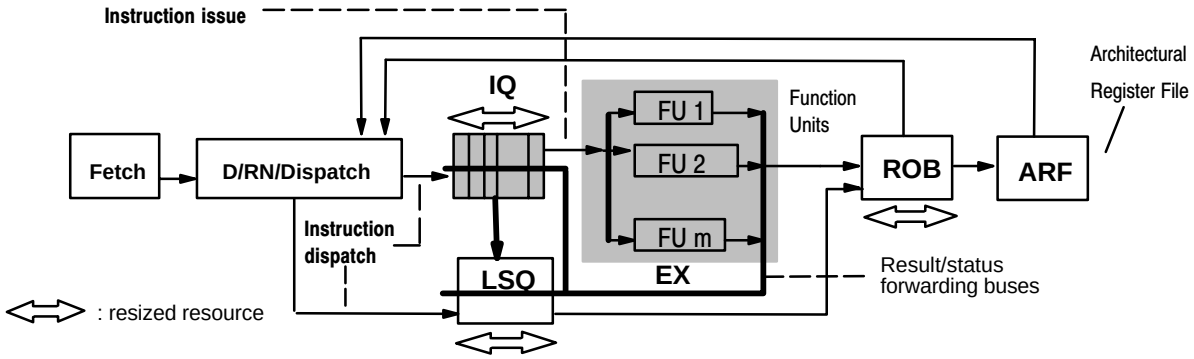


Figure 2. Superscalar datapath where ROB slots serve as physical registers

power with minimum performance loss was chosen for subsequent executions of the same hotspot. A 25% drop in the energy/instruction was reportedly achieved with this technique, as seen in the apparently preliminary results of [15]. Critical implementation details, including the overhead of hardware profiling and the manner in which optimum configurations are stored and reused were not discussed. The technique presented in this paper, in contrast, does not use dynamic profiling and instead adapts the datapath resources to match the demands of the program.

In [13], resource usages are controlled indirectly through pipeline gating and dispatch mode variations by letting the Operating System dictate the IPC requirements of the application. An industry standard, Advanced Configuration and Power Interface (ACPI) defining an open interface used by the OS to control power/energy consumption is also emerging [1]. Along similar lines, we are currently exploring a compiler–directed approach for dynamically allocating datapath resources based on the IPC requirements, using the basic infrastructure described in this paper.

In [4], a dynamic thermal management scheme was investigated to throttle power dissipation in the system by using several response mechanisms during the periods of thermal trauma. These included voltage and frequency scaling, decode throttling, speculation control and I–cache toggling. Some of these techniques indirectly controlled the resource usages but did nothing to vary the individual resource allocations. Consequently, they are of limited use in reducing leakage dissipations directly. Although not reported here, the technique proposed in this paper reduces such leakage dissipations, as it shuts down unused parts of the IQ, the LSQ and the ROB.

The reconfiguration of caches and the memory hierarchy in reducing the energy dissipation were explored in [2, 6] and represent approaches for reducing energy dissipation by matching the characteristics of some or all of the memory hierarchy to track the application's usage. Dynamic resource allocation within caches were studied in [6], [16] and [24]. More general forms of dynamically allocating cache and storage resources were proposed in [14]. As we focus on dynamic resource allocation directly connected with the instruction scheduling logic, we omit discussions on these techniques in this paper.

3. Superscalar Datapath and Sources of Energy Dissipation

Figure 2 depicts the superscalar datapath that we considered for this study. Here, the ROB entry set up for an instruction at the time of dispatch contains a field to hold the result produced by the instruction – this serves as the analog of a physical register. We assume that each ROB entry may hold only a 32–bit result, thus requiring the allocation of two ROB entries for an instruction producing a double–precision value. A dispatched instruction attempts to read operand values either from the Architectural Register File (ARF) directly (if the operand value was committed) or associatively from the ROB (from the most recently established entry for an architectural register), in case the operand value was generated but not committed. Source registers that contain valid data are read out into the IQ entry for the instruction. If a source operand is not available at the time of dispatch in the ARF or the ROB, the address of the physical register (i.e., ROB slot) is saved in the tag field associated with the source register in the IQ entry for the instruction.

When a function unit completes, it puts out the result produced along with the address of the destination ROB slot for this result on a forwarding bus which runs across the length of the IQ and the LSQ [19]. An associative tag matching process is then used to steer the result to matching entries within the IQ. Since multiple function units complete in a cycle, multiple forwarding buses are used; each input operand field within an IQ entry thus uses a comparator for each forwarding bus. Examples of processors using this datapath style are the Intel Pentium II and Pentium III. [18].

For every instruction accessing memory, an entry is also reserved in the LSQ at the time of instruction dispatch. As the address used by a load or a store instruction is calculated, this instruction is removed from the IQ, even if the value to be stored (for store instructions) has not yet been computed at that point. In such situations, this value is forwarded to the appropriate LSQ entry as soon as it is generated by a function unit. All memory accesses are performed from the LSQ in program order with the exception that load instructions may

bypass previously dispatched stores, if their addresses do not match. If the address of a load instruction matches the address of one of the earlier stores in the LSQ, the required value can be read out directly from the appropriate LSQ entry.

The IQ, the ROB and the LSQ are essentially implemented as large register files with associative addressing capabilities. Energy dissipation takes place in the issue queue in the course of: (a) establishing the IQ entries for dispatched instructions; (b) forwarding results from the FUs to the matching IQ entries, (c) issuing instructions to the FUs and, (d) flushing the IQ entries for instructions along the mispredicted paths.

Energy dissipations take place within the ROB during reads and writes to the register file that implements the ROB or when associative addressing is used. Specifically, these dissipations occur in the course of: (a) establishing the ROB entries, (b) reading out part of a ROB entry (when memory instructions are moved to the LSQ or when the valid data value for the most recent entry for an architectural register is read out), (c) reading out all of a ROB entry (at the time of committing an instruction), (d) writing results from FUs to the ROB entries and, (e) flushing the ROB entries on mispredictions or interrupts.

Energy dissipations occur within the LSQ in the course of: (a) establishing a LSQ entry, (b) writing computed effective addresses into a LSQ entry, (c) forwarding the result of a pending store in the LSQ to a later load, (d) forwarding the data in a register to be stored to a matching entry in the LSQ and (e) initiating D–cache accesses from the LSQ.

4. Simulation Methodology

The widely–used Simplescalar simulator [8] was significantly modified to implement *true hardware level, cycle–by–cycle* simulation models for such datapath components as the IQ, the ROB, the LSQ, register files, forwarding interconnections and dedicated transfer links. The Simplescalar simulator lumps the ROB, physical register files (PRFs) and the IQ together into RUUs, making it impossible to directly assess switching activities within these components and to make independent resizing decisions for each structure. Quite unlike the implementation used in Simplescalar (and power estimation tools like Wattch [3] and the one in [11], which are based on Simplescalar), the number of IQ entries and ROB entries – as well as the number of ports to these structures – are quite disparate in modern microprocessors, making it imperative to use distinct implementations of at least the ROB and the IQ in the simulator for accurately estimating the total chip power as well as the power dissipated by the individual components.

For estimating the energy/power for the key datapath components, the event counts gleaned from the simulator were used, along with the energy dissipations for each type of event described in Section 3, as measured from the actual VLSI layouts using SPICE. CMOS layouts for the IQ, the ROB and the LSQ in a 0.5 micron 4 metal layer CMOS process (HPCMOS–14TB) were used to get an accurate idea of the energy dissipations for each type of transition. The register files that implement the ROB, the IQ, and the LSQ were carefully designed to optimize the dimensions and allow for the use of a 300 MHz clock. A Vdd of 3.3 volts is assumed for all the measurements.

Parameter	Configuration
Machine width	4–wide fetch, 4–wide issue, 4–wide commit
Window size	32 entry issue queue, 128 entry reorder buffer, 32 entry load/store queue
L1 I–cache (I1–cache)	32 KB, 2–way set–associative, 32 byte line, 2 cycles hit time
L1 D–cache (D1–cache)	32 KB, 4–way set–associative, 32 byte line, 2 cycles hit time
L2 Cache unified	512 KB, 4–way set–associative, 64 byte line, 4 cycles hit time.
BTB/Predictor	1024 entry, 2–way set–associative, hybrid, gshare, bimodal
Memory	128 bit wide, 12 cycles first chunk, 2 cycles interchunk
TLB	64 entry (I), 128 entry (D), 4–way set–associative, 30 cycles miss latency
FUs and Latency (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 4 FP Add (2), 1FP Mult (4/1) / Div (12/12) / Sqrt (24/24)

Table 1. Architectural configuration of a simulated 4–way superscalar processor

Modern datapath implementations use more aggressive technologies compared to the 0.5 micron process used in this study. At small feature sizes, leakage power becomes significant – our dynamic resource allocation techniques will also reduce such leakage dissipations. Furthermore, with small feature sizes, the relatively higher contribution of wire capacitances also increases dynamic dissipations commensurately. Our dynamic resource allocation technique will also work better in this scenario, since the incremental resource allocation/deallocation mechanisms directly limit the impact of wire lengths by activating partitions selectively. The results reported in this paper are thus expected to be quite representative of what one would observe in implementations that use state–of–the–art process.

The configuration of a 4–way superscalar processor studied in this paper is shown in Table 1. In addition, we also performed the evaluation of a more aggressive 6–way machine. For the processor using a 6–way dispatch, 6–way issue and 6–way commit, we assumed the IQ and the LSQ of 64 entries, the ROB of 256 entries, I1–cache of 128 Kbytes, D1–cache of 64 Kbytes and L2 cache of 1 Mbytes. We also assumed that 6 integer ADD, 6 floating point ADD, 2 integer multiply/divide, 2 floating point multiply/divide and 3 Load/Store FUs were present. The latencies of FUs and the rest of the simulation parameters were unchanged compared to a 4–way machine from Table 1. For all of the SPEC 95 benchmarks, the results from the simulation of the first 200

million instructions were discarded and the results from the execution of the following 200 million instructions were used. Specified optimization levels and reference inputs were used for all the simulated benchmarks.

5. Resizing the Datapath Components: What Does It Take?

In this section, we discuss the hardware facilities needed to support incremental resource allocation and deallocation and the relevant constraints.

5.1. Multi–Partitioned Resources

The organization of the IQ allowing for incremental allocation and deallocation is depicted in Figure 3. The ROB and the LSQ are partitioned in a similar fashion. The IQ, the ROB and the LSQ are each implemented as a number of independent partitions. Each partition is a self–standing and independently usable unit, complete with its own precharger, sense amps and input/output drivers. Using separate prechargers and sense amps for each partition (as opposed to shared prechargers and sense amps for all of the partitions) makes it possible to use smaller, simpler and more energy–efficient sense amps and prechargers. Partitioning in this manner alone – independent of dynamic resource allocation – may result in some energy savings from the use of smaller and simpler sensing and precharging mechanisms, but this can be viewed as natural consequence of partitioning these resources for dynamic allocation.

A number of partitions can be strung up together to implement a larger structure. The connection running across the entries within a partition (such as bit–lines, forwarding bus lines etc.) can be connected to a common through line (shown on the right in Figure 3) through bypass switches. To add (i.e., allocate) the partition to the IQ and thus extend the effective size of the IQ, the bypass switch for a partition is turned on, and the power supply to the partition is enabled. Similarly, the partition can be deallocated by turning off the corresponding bypass switches. In addition, power to the bitcells in the shut–off partitions can be turned off to avoid leakage dissipation. We had no opportunity to observe the savings in leakage dissipation coming from dynamic resource allocation for the technology used in the current study, where leakage is extremely small. With smaller feature sizes and the use of lower supply voltages, leakage dissipation can be significant. Dynamic resource allocation, as introduced here, can reduce leakage dissipation by powering down unallocated partitions. Entries can be allocated in the IQ – across one or more active partitions in any order; an associative searching mechanism is used to look up free entries at the time of dispatching.

A subtle difference in shutting off segments of the bit–lines and the forwarding buses should be noted. For the forwarding bus lines, segments in *all currently active partitions* should be connected to the through line, as forwarding can take place to an entry within any one of the active partitions. In contrast, only the segment of a bit line *within the partition that contains an entry that is being read out or written to* have to be activated; there is no need to

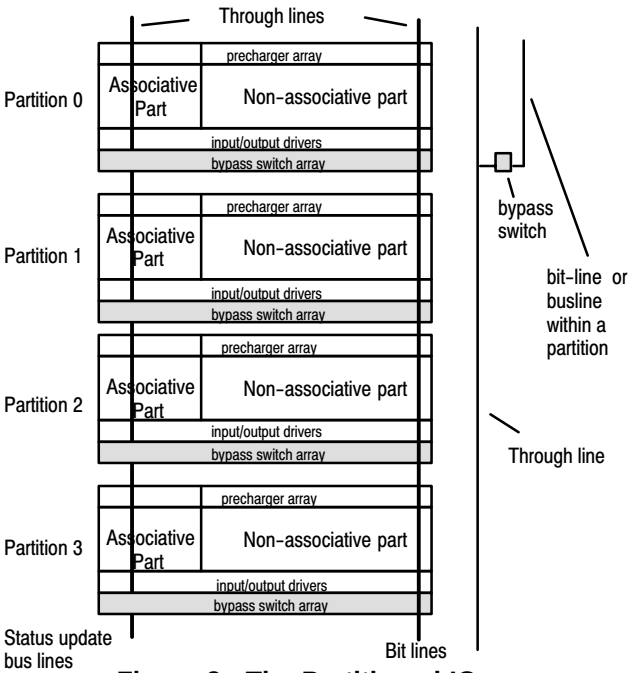


Figure 3. The Partitioned IQ

connect the bit–line segments of *all* active partitions to the through line. This latter technique, called bit–line segmentation [17], is routinely used within RAMs to reduce energy dissipation in the course of reads and writes. We will assume that bit–line segmentation is in use for the multi–partitioned resources.

The partition sizes (i.e., the number of entries within each partition) for the IQ, the LSQ and the ROB have to be chosen carefully. Making the partition sizes smaller allows for finer grain control for resource allocation and deallocation but small partition sizes can lead to higher partitioning overhead in the form of an increase in the layout area and a decrease in the energy savings. Optimal partition sizes thus exist for each of the resources under consideration. For the technology used, the optimal partition sizes are: 8 entries per partition for the IQ and the LSQ and a partition size of 16 entries for the ROB.

5.2. Resizing Constraints

The actual downsizing of the issue queue may not always be performed immediately after the resizing decision is made and is deferred till all instructions are issued from the IQ partition that is to be deactivated. The duration between the time a decision is made to downsize and the time of the actual deallocation of a partition is called a transition period. Instruction dispatching is blocked during the transition periods if the IQ entry allocated for the new instruction belongs to the partition that will become inactive. Otherwise, dispatches continue to the active IQ partitions.

The dynamic resizing of the ROB and the LSQ requires additional considerations because of the circular FIFO nature of these structures. We discuss these considerations below for the ROB.

The ROB is a circular FIFO structure with two pointers – the *ROB_tail* and the *ROB_head*. The *ROB_tail* indicates the next free entry in the ROB and is used during instruction dispatching to locate and establish the entries for the dispatched instructions. The *ROB_head* is used during the commit stage to update the architectural registers in program order. Figure 4 depicts a ROB with four partitions and also shows the some possible dispositions of the *ROB_head* and the *ROB_tail* pointers at the time the allocation/deallocation decision is made. The pointer CS (Current Size) indicates the current upper bound of the ROB. The pointer NS (New Size) specifies the newly established upper bound of the ROB after potential allocation or deallocation of one partition. The partition that is to be allocated/deallocated is shown as a dashed box.

To preserve the logical integrity of the ROB, in some situations partitions can be allocated and deallocated only when the queue extremities coincide with the partition boundaries. To deallocate a partition, two conditions have to be satisfied after the decision to downsize has been made. First, as in the case of the issue queue, all instructions from the partition to be deallocated must commit. Second, further dispatches must not be made to the partition being deallocated.

Deallocation scenarios are illustrated in Figure 4(a). In the situation shown in the top part, deallocation is delayed until the *ROB_tail* reaches NS and the *ROB_head* becomes zero. Notice, that the *ROB_head* wraps around twice before the deallocation can occur. This is, of course, an extreme case, as it is rare for the ROB occupancy to be very high immediately prior to the instant of reaching a downsizing decision. Slight variations of this specific case can also be considered, where the *ROB_tail* points to the left of NS. In that case, the *ROB_head* wraps around only once before the deallocation. The bottom part of Figure 4(a) is the case, where the deallocation of the partition marked by the dashed box can be performed immediately.

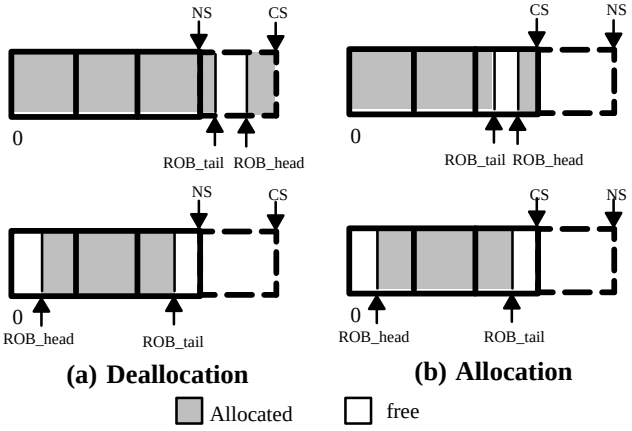


Figure 4. ROB resizing scenarios

To allocate a ROB partition, the value of the *ROB_head* pointer must be less than the value of the *ROB_tail* pointer to preserve the correct placement of the newly dispatched instructions into the ROB. Allocation scenarios are illustrated in Figure 4(b). The top part shows the situation where the allocation is deferred till the value of the *ROB_head* pointer reaches the value of zero. The bottom part shows the case where the allocation can be performed immediately. More details are given in [20].

6. Resource Usage in a Superscalar Datapath

We studied the correlations among the occupancies (number of valid entries) of the IQ, the ROB and the LSQ using our experimental setup. Representative results for one integer (*ijpeg*) and one floating point (*fpppp*) benchmark from the SPEC 95 suite are shown in Figures 5 and 6. Figures 5(a) and 6(a) show the occupancies of the three resources for *fpppp* and *ijpeg* benchmarks respectively, and Figures 5(b) and 6(b) show the ratios of these occupancies. Measurements were taken for 200 million committed instructions after skipping the first 200 million. For each benchmark, we recorded the average occupancies after every 1 million committed instructions and then graphed the results.

As seen from these graphs, the occupancies of the three resources are positively correlated. This suggests that resizing only one datapath resource is insufficient – in fact, if the sizes of other resources are not dynamically adjusted as well, these resources will remain overcommited most of the time. Another observation from Figures 5 and 6 is that it is difficult, if not impossible, to adjust the sizes of multiple resources by monitoring one of these resources and rescaling the number of active partitions within other resources proportionately. This is primarily because the ratios of the resource occupancies also change drastically across a program's execution. For example, for the *fpppp* benchmark, the ratio of the ROB occupancy to the IQ occupancy varies between 4 and 12 in the time interval shown (Figure 5(b)). One can consider resizing schemes, where the actual resource occupancies and their ratios are periodically sampled for a short duration and the appropriate prescaling coefficients are then set. However, this would increase the complexity of the control logic. For these reasons, we decided to independently monitor individual occupancies of the IQ, the ROB and the LSQ. The occupancy information was used to dynamically adjust the size of the IQ, the ROB and the LSQ independently. In the next section, we explain the details of our resizing strategy.

7. Resource Allocation Strategies

Two resizing strategies are used – one for downsizing a resource (turning off a partition) and the other for upsizing the resource (adding a partition). Our strategies are described for the IQ only; the strategies for the ROB and the LSQ are similar, provided that adequate considerations are made for the circular nature of the ROB and the LSQ, as discussed in Section 5.

7.1. Downsizing Strategy

The downsizing of the IQ is considered periodically – at the end of every IQ update period *(IQ_update_period*). During this period, the IQ occupancy, as measured by the number of allocated entries within the active partitions of the

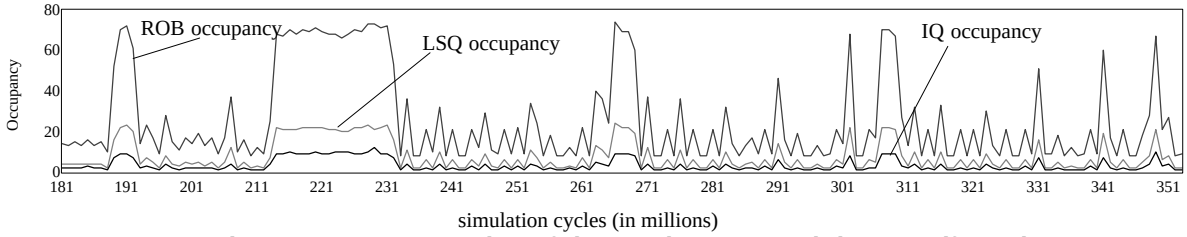


Figure 5a. Occupancies of the IQ, the ROB and the LSQ (fpppp)

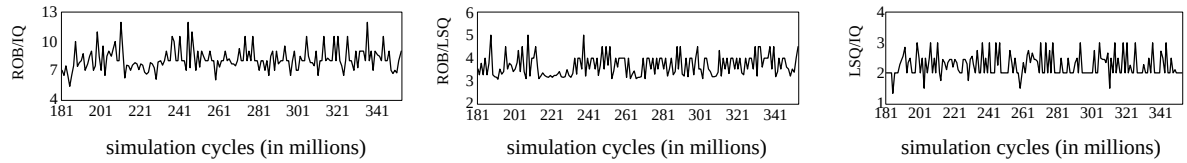


Figure 5b. Ratios between the occupancies of different datapath resources (fpppp)

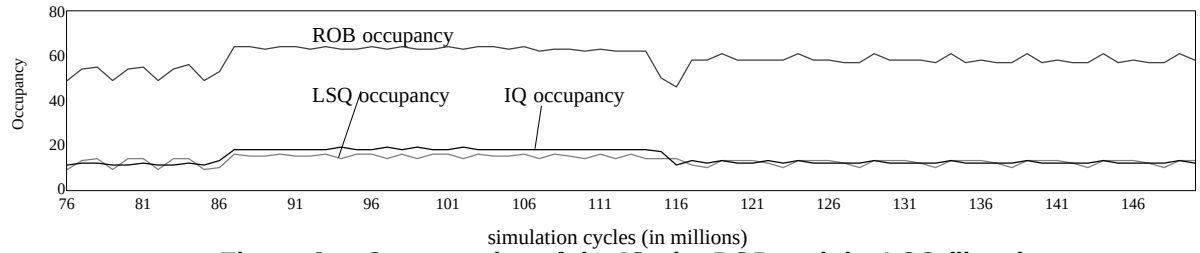


Figure 6a. Occupancies of the IQ, the ROB and the LSQ (ijpeg)

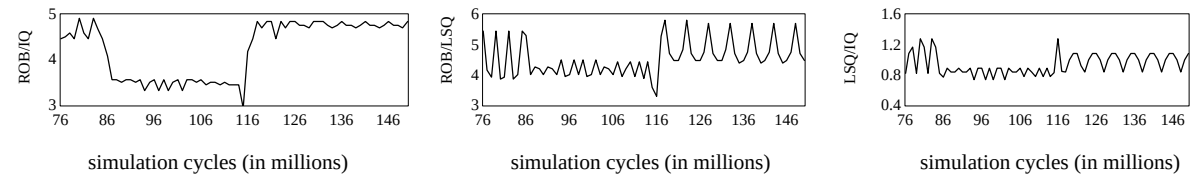


Figure 6b. Ratios between the occupancies of different datapath resources (ijpeg)

IQ, is sampled several times at periodic intervals. The value of the sampling interval *(IQ_sample_period*) is an integer divisor of *IQ_update_period*. The average of these samples is taken as the active IQ size (maintained in the variable (*active_IQ_size*) in the current update period. Both *IQ_update_period* and *IQ_sample_period* were chosen as powers of two to let the integer part and the fractional part of the computed IQ occupancy be isolated easily in the occupancy counter. This avoids the use of a full–fledged division logic. Figure 7 depicts the relationship between *IQ_update_period* and *IQ_sample_period*.

At the end of every IQ update period, we compute the difference, *diff=current_IQ_size–active_IQ_size*, where *current_IQ_size* is the size of the issue queue at that instant. If *diff* is less than the IQ partition size (*IQ_p_size*), no resizing is needed. If, however, *diff* is greater (or equal) than the IQ partition size, two scenarios are possible depending on whether an aggressive or a conservative downsizing strategy is implemented. In a conservative scheme, at most one IQ partition can be deactivated at a time. In an aggressive downsizing scheme, the maximum allowable number of partitions, *max_p*, as obtained directly from *diff* (*max_p=floor(diff/IQ_p_size*)), are deallocated.

The variable *active_IQ_size* provides a reasonable approximation of the average IQ occupancy within the most recent IQ update period. The difference between the current size of the IQ (*current_IQ_size*) and *active_IQ_size* in the update period indicates the degree of overcommited (that is, underused or unused) IQ partitions. The IQ is scaled down in size only if the sampling of the IQ occupancy indicates that the IQ partitions are overcommited. By thus downsizing resources only when resource partitions are overcommited we minimize the penalty on performance.

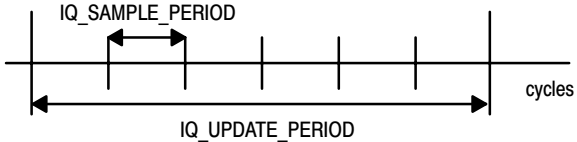


Figure 7. Sample period and update period used for the IQ downsizing

At the end of every IQ sample period, we record the IQ occupancy by using bit–vector counting logic for each active partition and then adding up these counts. The fact that this counting is performed only at the end of the sampling period (as opposed to maintaining a dynamic count on a cycle–by–cycle basis) suggests that the energy dissipated in estimating the IQ occupancy is tolerably small. A rough estimation indicates that the monitoring/counting logic expends less than 1% of the total power dissipated in the IQ, the ROB and the LSQ combined.

7.2. Upsizing Strategy

The resizing strategy implemented to scale the IQ size back up once the application begins to demand more resources effectively uses the rate at which dispatches block due to the non–availability of entries within the IQ. We use the IQ overflow counter to count the number of cycles for which dispatch is blocked because a free entry is not available in the IQ. This counter is initialized to 0 at the beginning of every IQ update period and also immediately after making additional resource allocations within the IQ. The reason for resetting the counter in the latter instance is described in Section 7.3. Once the value of this counter exceeds a predetermined IQ overflow threshold, (maintained in the variable *IQ_overflow_threshold*), one of the currently deactivated IQ partitions, if any, is turned on, effectively upsizing the IQ. Note, that additional resource allocations can take place well before the expiration of an IQ update period, to react quickly to the increased demands of the program. The upsizing strategy is thus more aggressive than the downsizing strategy.

7.3. Other Considerations

The main reason for taking multiple samples within a single update period is to get a better estimate of the average resource usage within this update period. An upsizing within an update interval can distort this average quite drastically and result in an erroneous downsizing at the end of this update period. Such a situation causes instability in the control mechanism used for our scheme. It is for this reason that we abandon the current update period and start a new update cycle immediately after resizing up.

For similar reasons, during the transition periods immediately prior to an actual downsizing, no statistics are collected (all counters controlling resizing are frozen at 0).

By varying the values of *IQ_overflow_threshold* and *IQ_update_period*, either of the upsizing or the downsizing processes can be made more aggressive depending on the design goals as dictated by power (downsizing) / performance (upsizing) tradeoffs.

8. Results and Discussion

To estimate the power savings effects of the resizing mechanism proposed in this paper, we evaluated three configurations of the IQ, the ROB and the LSQ. These are as follows:

The base case, B: Here, each resource is a monolithic structure, as used in traditional datapath designs.

Base case with bit–line segmentation, S: Here, the base case (B) is augmented with the bit–line segmentation [17] – a well–known power savings technique designed to reduce the effective bit–line capacitance. The size of each segment parallels the partition sizes chosen for dynamic allocation. We analyzed this configuration because we wanted to isolate the effects of bit–line segmentation – a natural technique to be used with the partitioned organization and resizing. On the average across all SPEC 95 benchmarks, the organization S saves 13.2% in power dissipation within the IQ, 37.1% within the ROB, and 8.7% within the LSQ as compared to the organization B.

Partitioned organization with resizing, R: Here, the IQ, the ROB and the LSQ are implemented as shown in Figure 3 and as described in Section 5.

Our experiments showed that the behavior of our resizing scheme is most sensitive to the changes in the overflow thresholds. Table 2 shows performance degradation caused by resizing in the form of drops in the IPC. Also shown in this table are power savings and average active sizes (number of entries that are turned on) of the IQ, the ROB and the LSQ for various values of *overflow_threshold*, common to all these components. Results are averaged over all SPEC95 benchmarks. For these simulation runs, we fixed the value of *update_period* for all three resources at 2K cycles. This value has to be chosen in an optimal fashion. Making it too large would cause variations in the resource usages to go unnoticed. Making it too small would result in a prohibitive resizing overhead. We also studied the effects of various update periods on our resizing scheme and results are presented later in this section. The value of *sample_ period* was chosen as 32 cycles, allowing for the acquisition of 64 occupancy samples in one update period. In this and the following tables, OT stands for "overflow threshold", UP stands for "update period", PS stands for "power savings", and "size" specifies the average active size of the IQ, the ROB or the LSQ as measured by the number of activated entries.

OT	IPC drop (%)	IQ size	ROB size	LSQ size
128	0.58	23.0	78.7	22.9
256	1.89	20.2	67.6	20.2
512	4.86	17.2	56.8	17.5
1024	9.63	14.2	46.4	14.6
2048	13.97	11.9	37.6	12.3

Table 2. Average number of active entries within the IQ, the ROB and the LSQ and performance degradation for various overflow thresholds

As seen from the data in Table 2, lower values of *overflow_threshold* result in virtually no performance loss, but the average number of turned–on entries and thus, power savings potential, are limited. For larger values of *overflow_threshold*, the potential power savings can be significant, but this comes at a cost of noticeable performance drop – as high as 14%. We did not perform any experiments for the values of *overflow_threshold* less than

128, because the average performance loss was already well below 1% in this case. Decreasing the value of *overflow_threshold* further would have an adverse effect on energy dissipation (more partitions will be on) with no further performance improvement. The maximum possible value of *overflow_threshold* for a resource is equal to the value of *update_period* used for that resource (2048 in our experiments). In this case, the resource upsizing is triggered if, during the update period, resource overflows are encountered every cycle. Thus, we effectively exhausted the space of all reasonable values of overflow threshold for a given update period (considering, of course, that these values are powers of two for implementation reasons, as discussed above). Our experiments showed that the overflow threshold value of 512 gave the optimal power/performance trade–off for the majority of the simulated benchmarks. Table 3 shows the performance of the resizing scheme for individual benchmarks for OT=512. Power savings are given as an improvement of the R configuration over the B configuration.

Higher power savings are achieved for the integer benchmarks, because they are generally less resource–hungry than the floating point benchmarks and present more opportunities for downsizing. For the IQ, the lowest savings are for *hydro2d* and *mgrid* benchmarks – these use the full IQ most of the time. For the majority of integer benchmarks, only 2 of the 4 available partitions are predominantly turned on, resulting in power savings of about 48% on the average. Notice also, that power savings are not always proportional to the active size of the resource.

Benchmark	IPC drop%	IQ size	IQ PS%	ROB size	ROB PS%	LSQ size	LSQ PS%
compress	+2.5	14.7	45.8	40.6	61.9	8.0	67.2
vortex	8.3	12.1	48.3	33.8	67.1	20.4	42.8
m88ksim	3.6	16.3	42.4	30.8	64.7	14.3	48.4
gcc	3.3	12.1	50.4	25.6	68.5	12.9	52.6
go	4.9	13.4	51.0	27.1	69.4	10.9	52.4
ijpeg	6.5	19.9	43.5	48.1	65.8	16.9	37.4
li	3.8	14.1	47.3	30.6	65.8	14.9	48.8
perl	0.0	13.9	46.8	36.5	63.3	18.2	41.1
turb3d	9.2	20.9	45.9	61.0	63.7	14.8	49.9
fpppp	0.9	11.2	42.5	38.6	61.3	14.4	40.7
apsi	2.6	25.1	32.0	115.4	45.5	28.2	14.9
applu	14.8	16.0	50.5	65.9	64.0	16.2	41.0
hydro2d	2.2	25.0	29.3	123.4	41.0	26.9	12.4
mgrid	0.4	27.0	25.2	122.1	42.2	30.7	12.4
su2cor	8.0	14.0	49.4	29.8	67.2	12.8	51.3
swim	2.0	20.6	36.0	89.1	48.1	27.6	17.0
tomcatv	7.0	14.3	48.2	37.9	67.1	12.4	58.1
wave5	12.6	18.6	47.4	66.9	66.7	14.4	58.5
Int Avg.	3.5	14.6	46.9	34.1	65.8	14.6	48.8
FP Avg.	6.0	19.3	40.6	75.0	56.7	19.8	35.6
Average	4.9	17.2	43.4	56.8	60.7	17.5	41.5

Table 3. Behavior of SPEC 95 benchmarks for overflow_threshold=512 and update_period=2048

Sometimes, power savings may be higher for the benchmarks with more active partitions (*applu*) than for the benchmarks with fewer active partitions (*fpppp*). This is because of two reasons. First, the IPC drop of *applu* is very high (in excess of 14% – the highest of all simulated benchmarks), resulting in a longer execution time and the reduction of energy dissipated per cycle compared to the base case. The same argument can be made for the dispatch and issue rates. Second, power savings depend on the number of comparators that are used during forwarding. In our power estimations, we assumed that only the comparators associated with the invalid slots within valid IQ entries (in other words, slots waiting for a result) are activated during forwarding. The average number of such activated comparators and the percentage of power dissipated in this process varies for different benchmarks. Resizing, as used in the context of segmented bit–lines, saves relatively less power during forwarding than during non–associative reads/writes at the times of issue/dispatch. Thus, the power savings achieved within the resizable IQ is not only dependent on the size of the active portion of the IQ, but also is a function of the IPC, dispatch rate and and program's characteristics.

Similar observations are applicable for the ROB and the LSQ. The percentage of power savings is higher for the ROB, because dissipations due to the associative addressing are relatively smaller for the ROB than for the IQ and the LSQ. Note that for all of the benchmarks except for *applu* and *wave5*, performance loss because of dynamic resizing is below 10%, with the average being less than 5% across the entire benchmark suite. We observed a slight performance improvement for *compress95*. This is because *compress95* has a low branch prediction accuracy and a smaller issue window reduces the amount of work done along mispredicted path. For different configurations, we observed a similar phenomenon for *li* and *perl*. Notice that even when resources are used to their full extent, substantial power savings are still achieved – this is, of course, the result of bit–line segmentation and the use of multi–partitioned resources, as discussed in Section 5.1.

Our experiments showed that the best tradeoff between power and performance is achieved when the ratio of update period to overflow threshold is 4 for each resource (Table 3 represents one of these cases). Here, resource upsizing is triggered when the frequency of overflows is more than 1 in 4 cycles. We studied the effects of various update periods on the performance of our resizing mechanism, keeping the ratio of update period to overflow threshold at 4. Some of these results, averaged across all benchmarks, are summarized in Table 4. In all cases, the sample period was set to allow for the acquisition of 64 samples during one update period.

Larger update periods result in higher performance loss, because of the slower reaction of the upsizing phase to the changes in the application's demands. This is, of course, the consequence of higher overflow thresholds which have to increase commensurately with the update periods to maintain the same level of tradeoff between power and performance. The advantage of larger update periods is the

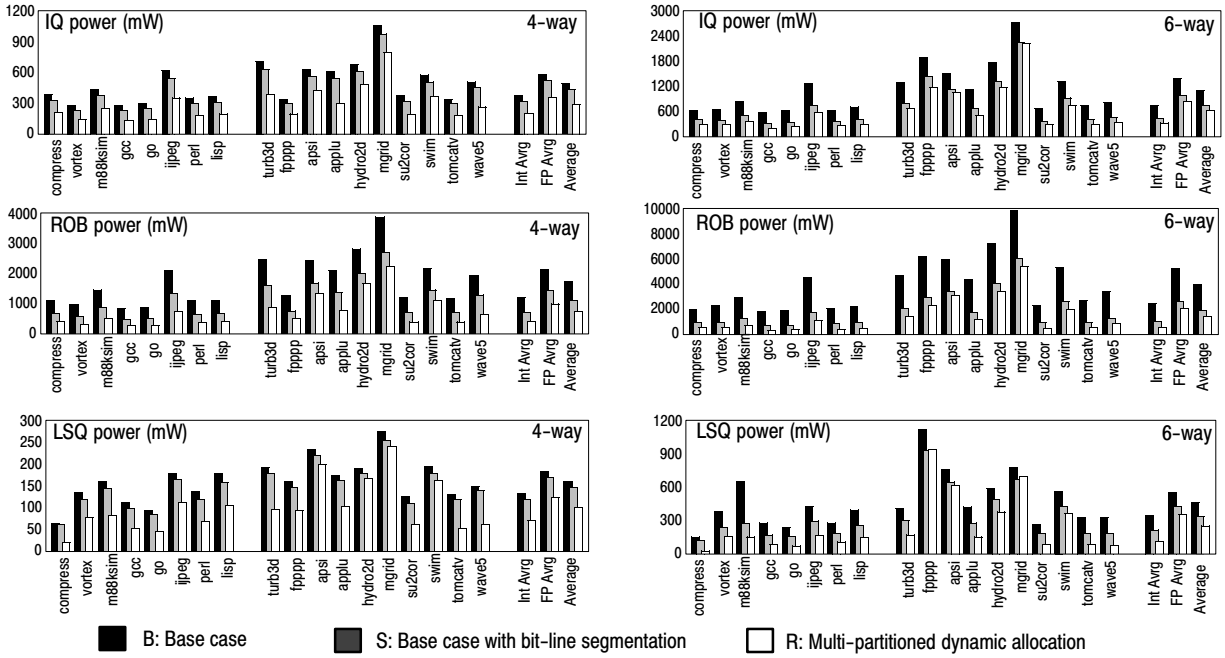


Figure 8. Power dissipation in the IQ, ROB and LSQ for 4–way and 6–way processors

reduction of control logic overhead, because resource monitoring and resizing decisions are made at a coarser granularity. The differences in terms of power savings achieved for update periods studied are quite small, as indicated in Table 4 by the average active sizes.

To see the effects of our resizing scheme on power dissipation in more aggressive superscalar designs, we also simulated a 6–way processor with larger sizes of the ROB, the IQ, and the LSQ (as discussed in Section 4). As expected, the results show a higher percentage of power reduction because the degree of resource overcommitment is higher for a 6–way processor. The higher power savings for a 6–way machine come at the cost of a higher performance drop, which is also not surprising because the base case IPCs of programs executed on a 6–way machine are higher than for a 4–way machine. For the common overflow threshold of 512, the average performance loss across all SPEC 95 benchmarks was observed as 7.3% with the average power savings of 50% in the IQ, 70.5% in the ROB and 55.7% in the LSQ. Total power savings across all three components amounts to 59% on the average.

UP, OT	IPC drop %	IQ size	ROB size	LSQ size
128, 32	4.31	17.3	57.2	17.6
2048, 512	4.86	17.2	56.8	17.5
8192, 2048	5.13	17.1	56.4	17.4

Table 4. Power savings and performance degradation for various update periods

Figure 8 summarizes power savings achieved by resizing technique described in this paper for the 4–way and 6–way machines. Results are shown for the overflow threshold value of 512, update period value of 2048 and sample period of 32 cycles for all structures. Power savings are shown for the B, S and R configurations. Notice, that for some of the benchmarks (*mgrid* and *fpppp*), configuration S results in higher power savings within the LSQ than configuration R for 6–way machine. This is because the entire LSQ is used throughout the execution of these benchmarks on the 6–way processor, and the increase in power consumption stems mainly from the dissipations associated with the segmented forwarding buses.

Finally, we studied the use of aggressive downsizing mode. The aggressive mode saves less than 8% in the average ROB size, and less than 2% in the average IQ size. The difference is almost invisible for the SPEC95int benchmarks. The SPEC95fp benchmarks have some benefit from the aggressive mode, as discussed in Section 7.1, with some penalty in the IPC. The IPC difference between these two modes is less than 2% in favor of non–aggressive mode.

9. Concluding remarks

The "one–size–fits–all" philosophy in allocating datapath resources results in the resource overcommitment. Our approach to minimizing the power requirements of the datapath is to use a dynamic resource allocation strategy that tries to closely track the actual dynamic resource demand of the executing program. We primarily focused on using simple techniques to resize the IQ, the ROB (integrating physical registers) and the LSQ independently. In particular, these components are partitioned and the number of active (i.e., powered up) partitions are chosen dynamically to closely track the actual demands of the program. The IQ, the

LSQ and the ROB are controlled independently. Downsizing is driven by directly using sampled estimates of their individual occupancies. Upsizing, on the other hand, is done more aggressively based effectively on the rate at which dispatches block due to the lack of these individual resources. Our power savings technique is technology independent and can be used in conjunction with other orthogonal techniques for saving power.

Our results show that the error arising from the computations of the average occupancy by sampling the actual occupancies at discrete points is tolerable since significant energy savings are achieved using our approach with very little impact on performance. For a 4–way superscalar CPU, an average power savings of 52.6% is achieved across the SPEC 95 benchmarks for the IQ, LSQ and ROB combined, with an average performance penalty of only 4.86%. For a 6–way dispatch rate, the power savings on the average are 59.3% across these structures, with an average performance penalty of 7.3%. Thus, our technique can save a significant amount of power with a tolerable performance loss.

Although the current paper shows the savings on dynamic/switching power, dynamic deallocation of partitions also saves leakage power that would be otherwise dissipated within the IQ, the ROB and the LSQ.

As an extension of the work described in this paper, we are currently exploring the control and dynamic allocation of datapath resources beyond the IQ, the ROB and the LSQ (such as register files, caches, function units). Additional work in progress is looking at the use of compiler–inserted directives in unused fields of instructions to change resource allocation dynamically – a natural transition of the hardware–directed resource allocation techniques of the current work into the compiler.

10. References

[1] Advanced Configuration and Power Interface Specification (Intel, Microsoft, Toshiba), 1999.

[2] Albonesi, D., "Selective cache ways: On–demand cache resource allocation", *in Proc. of the Int'l Symposium on Microarchitecture*, 1999.

[3] Brooks, D., Tiwari, V., Martonosi, M., "Wattch: A Framework for Architectural–Level Power Analysis and Optimizations", *in Proc. of the 27th Int'l Symposium on Computer Architecture*, 2000.

[4] Brooks, D., Martonosi, M., "Dynamic Thermal Management for High–Performance Microprocessors," *7th Int'l Symposium on High–Performance Computer Architecture (HPCA–7)*, 2001.

[5] Bahar, I., Manne, S., "Power and Energy Reduction Via Pipeline Balancing", *in Proc. of the Int'l Symposium on Computer Architecture*, 2001, pp.218–229.

[6] Balasubramonian, R., Albonesi, D., Buyuktosunoglu, A., and Dwarkadas, S., "Memory Hierarchy Reconfiguration for Energy and Performance in General–Purpose Processor Architectures", *in Proc. of the 34rd Int'l Symposium on Microarchitecture*, 2000.

[7] Bhandarkar, D., "Alpha Implementations and Architecture Complete Reference and Guide", Digital Press, 1996.

[8] Burger, D., and Austin, T. M., "The SimpleScalar tool set: Version 2.0", Tech. Report, Dept. of CS, Univ. of Wisconsin–Madison, June 1997 and documentation for all SimpleScalar releases (through version 3.0).

[9] 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 Proc. of Great Lakes Symposium on VLSI Design*, 2001.

[10] Buyuktosunoglu, A., Schuster, S., Brooks, D., Bose, P., Cook, P. and Albonesi, D., "An Adaptive Issue Queue for Reduced Power at High Performance", *Workshop on Power–Aware Computer Systems*, held in conjunction wth ASPLOS, November 2000.

[11] Cai, G., "Architectural Level Power/Performance Optimization and Dynamic Power Estimation", *in Proc. of the Cool–Chips tutorial.* An Industrial Perspective on Low Power Processor Design in conjunction with MICRO–32, 1999.

[12] Folegnani, D., Gonzalez, A., "Energy–Effective Issue Logic", *in Proc. of the Int'l Symposium on Computer Architecture*, 2001, pp.230–239.

[13] Ghiasi, S., Casmira, J., and Grunwald, D., " Using IPC variation in workloads with externally specified rates to reduce power consumption", *in Proc. of the Workshop on Complexity–Effective Design*, June 2000.

[14] Huang, M., Renau, J., Yoo, S–M. and Torellas, J., "A Framework for Dynamic Energy Efficiency and Temperature Management", *in Proc. of the 33rd Int'l Symposium on Microarchitecture*, 2000.

[15] Iyer, A. and Marculescu, D., "Run–time Scaling of Microarchitecture Resources in a Processor for Energy Savings", *in Proc. of Kool Chips Workshop*, held in conjunction with MICRO–33, December 2000.

[16] Kaxiras, S., Hu, Z. and Martonosi, M., "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power", *in Proc. of the Int'l Symposium on Computer Architecture*, 2001, pp.240–251.

[17] Kucuk, G., Ghose, K., Ponomarev, D., Kogge, P., "Energy Efficient Instruction Dispatch Buffer Design for Superscalar Processors", *in Proc. of Int'l Symposium on Low–Power Electronics and Design*, 2001, pp.237–242.

[18] Microprocessor Report, various issues, 1996–1999.

[19] Palacharla, S., Jouppi, N. P. and Smith, J. E., "Quantifying the complexity of superscalar processors", Technical report CS–TR–96–1308, Dept. of CS, Univ. of Wisconsin, 1996.

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

[21] Sherwood, T. and Calder, B., "Time Varying Behavior of Programs", Tech. Report No. CS99–630, Dept. of Computer Science and Engg., UCSD, August 1999.

[22] Wall, D.W., "Limits on Instruction Level Parallelism", *in Proceedings of ASPLOS*, November 1991.

[23] Wilcox, K., Manne, S., "Alpha processors: A History of Power Issues and a Look to the Future", *in Cool–Chips Tutorial*, November 1999.

[24] Yang, S–H. Powell, M. D., Falsafi, B., Roy, K. and Vijaykumar, T.N., "An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep–Submicron High–Performance I–caches", *in Proc. of Int'l Conference on High Performance Computer Architecture*, 2001.

[25] Zyuban, V. and Kogge, P., "Optimization of High–Performance Superscalar Architectures for Energy Efficiency", *in Proc. of Int'l Symposium on Low–Power Electronics and Design*, 2000, pp. 84–89.