Energy-Effective Issue Logic

Daniele Folegnani and Antonio González

Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Jordi Girona, 1-3 Mòdul D6
08034 Barcelona, Spain
antonio@ac.upc.es

Abstract

The issue logic of a dynamically-scheduled superscalar processor is a complex mechanism devoted to start the execution of multiple instructions every cycle. Due to its complexity, it is responsible for a significant percentage of the energy consumed by a microprocessor. The energy consumption of the issue logic depends on several architectural parameters, the instruction issue queue size being one of the most important. In this paper we present a technique to reduce the energy consumption of the issue logic of a high-performance superscalar processor. The proposed technique is based on the observation that the conventional issue logic wastes a significant amount of energy for useless activity. In particular, the wake-up of empty entries and operands that are ready represents an important source of energy waste. Besides, we propose a mechanism to dynamically reduce the effective size of the instruction queue. We show that on average the effective instruction queue size can be reduced by a factor of 26% with minimal impact on performance. This reduction together with the energy saved for empty and ready entries result in about 90.7% reduction in the energy consumed by the wake-up logic, which represents 14.9% of the total energy of the assumed processor.

Keywords: Issue logic; energy consumption; low power; adaptive hardware.

1. Introduction

Power consumption has become an important concern in processor design. More than 95% of current microprocessors produced today are used in embedded systems, so low power requirements are nowadays critical[30]. Mobile systems need an extremely efficient use of the energy due to their limited battery life, which is not expected to experience drastic increases in the near future. On the other hand, high performance microprocessors have to deal with heat dissipation and high current peak problems. This impose very expensive cooling systems with an increasing relative cost with respect to the total chip cost. For actual cooling systems and technology, beyond 50 Watt of total dissipation, the cost of dissipating a further Watt becomes superlinear, and may soon reach an unreasonable cost [27]. Besides, several failure mechanisms such as thermal runaway, gate dielectric, junction fatigue, electro-migration diffusion, electrical-parameter shift and silicon interconnections fatigue become significantly worse as temperature increases [28]. As an example of the current trend in power consumption, current microarchitectures such as the Alpha 21364 and PowerPC 704 dissipate about 85 and 100 Watt respectively [31]. Another important question is that the growing demand of multimedia functionalities for computer systems [16] require an increasing computing power, which sometimes is achieved through higher clock frequencies and more sophisticated architectural techniques, with an obvious impact on power consumption.

With the fast increase in design complexity and reduction in design time, new generation CAD tools for VLSI design like PowerMill [13] and QuickPower [12] are crucial to evaluate the energy consumption at different points of the design, helping to make important decisions early in the design process. The importance of a continuous feedback between the functional system description and the power impact requires better and faster evaluation tools and models to introduce changes rapidly through various design scenarios. Recent research has shown how architectural power estimation tools like SimplePower [36], Wattch [6] and Architectural Power Evaluation [7] can accurately estimate the power consumption of a whole microprocessor with a reasonable overhead and accuracy.

In this paper we first evaluate the energy consumed by the different parts of a superscalar processor through the Architectural Power Evaluation tool [7], which is a detailed cycle-level simulator that includes both performance and power consumption estimation techniques. This analysis demonstrates that one of the critical components for power consumption in modern superscalar processors is the part devoted to extract parallelism from applications at run time. In particular, the wake-up function, which analyzes data dependencies of the program through an associative search in the instruction queue, is the main power consuming part of the issue logic. This is reflected in a high complexity circuit and high logic activity. For a typical microarchitecture the instruction queue and its associated issue logic are responsible for about 25% of the total power consumption on average. This observation motivates an analysis of the effectiveness of a large instruction queue.

We first note that both empty entries and ready entries consume energy for doing a useless activity: the former do not contain any valid data whereas the latter are known to be ready, so they do not need to be checked by the wake-up logic. Besides, we observe that the instruction queue must be large if its size is fixed and high performance is desired for a wide range of applications. However, different applications may require different queue sizes, as already pointed out by other authors [2]. In addition, the queue size requirements may significantly vary during the execution of a program. Based on these observations, we propose a technique to dynamically resize the instruction queue according to the characteristics of the dynamic instruction stream. This technique reduces the effective size of the instruction queue by about 26% on average. We show that these mechanisms reduce the energy

consumption of the wake-up logic by about 90% with respect to a fixed size instruction queue, while the performance is hardly affected.

The rest of this paper is organized as follows. Section 2 reviews previous work. Section 3 describes the experimental framework. Section 4 analyzes the energy consumption of a typical superscalar processor. Section 5 presents the proposed techniques to reduce energy consumption and section 6 evaluates their performance. Finally, section 7 summarizes the main conclusions of this work.

2. Related work

A large number of techniques for low power have been proposed in the literature. The problem has been attacked from different standpoints, ranging from a high-level perspective, such as the operating system, to the circuit and technology level. Two main objectives of the low power research area are the peak power reduction, (i.e. reducing the maximum power dissipated by the die), and the total energy reduction for a given workload.

Power consumption in CMOS technology is quadratic with voltage supply and linear with switching capacitance and activity. Orchestrated methods to scale voltage and frequency simultaneously can be really effective to drastically reduce the power requirements of microarchitectures. As examples, Intel StrongARM SA-110 [11] and Transmeta Crusoe [14] use combinations of these techniques to considerably reduce power consumption.

Another example used in commercial products is the TAU (Thermal Assist Unit) of the PowerPC G3 and G4 family, where an interrupt is generated based on a programmable thermal threshold [25]. This causes an instruction flow reduction between instruction cache and the instruction buffer in order to decrease the temperature of the chip. Typically, voltage and frequency scaling or interruption-based power-aware techniques are managed by the operating system or the applications. Recent studies [5] have investigated adaptive thermal management and power-conscious dynamically reconfiguration engines for high-performance microprocessors.

Today's CPU designers also use conditional clocking or clock gating to disable parts of the architecture when their activity is not useful [6].

A comprehensive evaluation of the trade-offs between power and performance of different architectural paradigms can be found in [18]. In that work, the authors study the effect of different architectural models on power consumption, starting from a non pipelined to an aggressive superscalar architecture.

Many research works have focused on reducing power consumption in cache memories [29][1][19][20], which is a critical component for power consumption, especially in the embedded microprocessor world.

Other works have proposed to reduce the speculative activity of a processor in order to avoid useless activity. For instance, by means of a confidence estimator for branch predictions, the processor may decide not to speculate on branches that are hard to predict [21].

Albonesi presents a study of the influence of the size of several architectural structures on IPC, clock rate and the power requirements [3][2]. Marculescu proposes a mechanism to dynamically adapt the fetch and execution bandwidth based on a profiling at the basic-block level [22].

Code and data compression [24][15][8] can reduce activity in memories and buses although they introduce the overhead of decompressing tasks.

Some compiler techniques can also be found in the literature. For instance, instruction scheduling approaches [9] can be used to avoid large current peaks in the execution core of high performance processors.

3. Experimental Framework

In this section we present the framework used to estimate the performance and power consumption of a typical superscalar processor. For this purpose, we have used the Architectural Power Evaluation tool [7] based on the SimpleScalar simulator [4], with extensions to compute the energy spent every cycle.

3.1. Power Consumption Estimation Model

The Architectural Power Evaluation tool [7] considers the processor microarchitecture partitioned into 32 blocks, each corresponding to a basic functional block and a circuit block. The internal connections of blocks are modeled as components of the blocks. The power consumption of the interconnections among blocks is not considered since it is usually a negligible part of the total power consumption. Every block is divided in subparts, depending on the hardware implementation, and the simulator counts the number of times that each subpart is used along the program execution. The power consumption of each subpart is characterized by the following three parameters: power density, area occupied and circuit type. Five different types of circuits are considered: memory, static logic, dynamic logic, PLA and clock distribution[1].

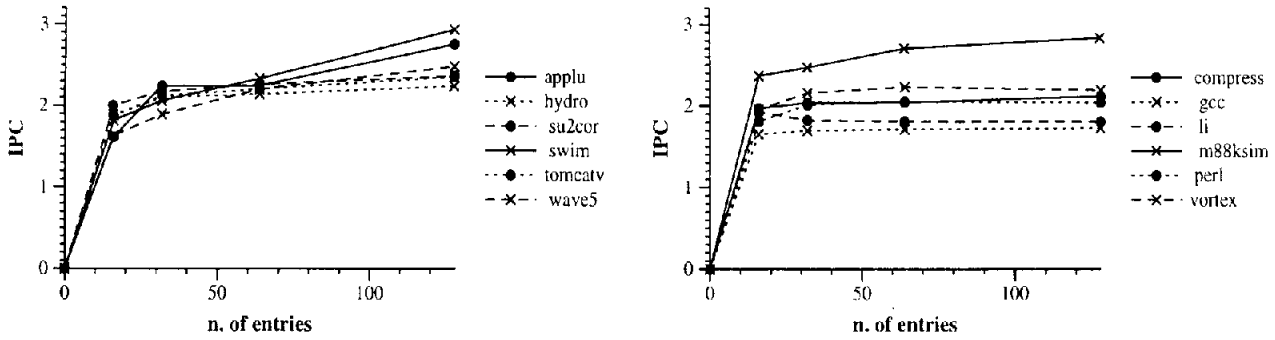
In the model, the power characterization of a digital synchronous functional block of the microarchitecture is based on the assumption that the power spent in a given cycle is proportional to the activity of the block, the power density of the block and the given area. The power computation of a given block j in a given cycle i is computed through the following expression:

$$\sum_k \alpha_k[i] A_k \times APD_k + \sum_k (1 - \alpha_k[i]) \times A_k \times IPD_k$$

Where k represents each subpart of block j; α is the activity of each subpart (activity is represented by the value 1 and inactivity by 0); A, APD and IPD are the area, the active power density, and the inactive power density of each subpart respectively. The power density parameters are usually calculated from SPICE circuit simulations. The area of the different blocks can be obtained from soft core vendors and from VLSI textbooks. Well-known VLSI scaling techniques can be applied to extrapolate area and power density for newer process technologies.

In this process we assume a 0.18 µm CMOS technology. Other physical characteristics of the different blocks of the architecture can be found elsewhere [7] [17].

1. Note that clock distribution, which is one of the main sources of consumption, is included in the consumption of each block



IPC vs window size for FP programs

IPC vs window size for INT programs

Figure 1: IPC for different instruction window sizes.

This kind of architecture partitioning and modeling ensures a good power estimation accuracy at architectural level, allowing architects to have a quite detailed power map of the microarchitecture. Low level design tools usually compute power consumption with excellent accuracy but require the complete HDL description and the circuit design. Furthermore, the simulation cost is a few orders of magnitude bigger compared to an RTL simulator, being not acceptable for exploring a high-complexity circuit such as a whole superscalar microprocessor, and given the typically large architectural design space.

3.2. Benchmarks

For this study we have randomly selected a subset of the Spec95 benchmarks (applu, swim, tomcatv, wave, su2cor, hydro2d, perl, li, m88ksim, vortex, compress and gcc). We have simulated 100 million of instructions after skipping the initial 100 million of instructions for each of them. The benchmarks were compiled with the Compaq/Alpha compiler with -O4 optimization flag.

3.3. Architectural Model and Parameters

A typical superscalar microarchitecture is considered in this work. The main parameters of the microarchitecture are described in Table 1. We assume an instruction window composed of an instruction queue and a reorder buffer that are organized as FIFO queues. Decoded instructions are inserted in both queues at dispatch time in program order. Instructions leave the instruction queue when they are issued and free their reorder buffer entries when they commit. In each entry, the instruction queue contains the operation code and the instruction source operands if available or the tags that uniquely identify them otherwise. It also contains a ready bit for each tag and a valid entry flag. Each reorder buffer entry holds the instruction program counter, the destination-register operand tag, and a bit that indicates whether the instruction has been executed. Program counter is used to maintain precise exception and the tag is used to identify the destination operand. We assume a microarchitecture where the results of instructions are kept in the reorder buffer until they are committed. Then, they are copied into the architectural register file. This is the approach used by some commercial microprocessors such as the Pentium III [10].

Parameter	Configuration
Fetch width	4 instructions
I-cache L1	128KB, direct mapped, 32 byte line, 1 cycle hit time, 3 cycle miss penalty
Branch Predictor	Hybrid with 1K entry Gshare, 8bit global history, 2K entry bimodal and 1K entry selector
Decode/Rename and Retire width	4 instructions
Instruction queue size	128
Functional units	4 intALU,4 fpALU 1 int mul/div, 1 fp mul/div
Issue Mechanism	Out of order issue, store forwarding oldest ready first selection policy
D-cache L1	128KB, 4-way set associative, 32 byte line, 1 cycle hit time, 3 cycle miss penalty
I/D-cache L2	1MB, 4-way set associative, 64 byte line, 3 cycle hit time, 64 bytes bus bandwidth to main memory, 16 cycles first chunk, 2 cycles interchunk

Table 1: Architectural configuration

The number of entries of the instruction queue and reorder buffer are assumed the same for the baseline configuration and is experimentally determined in order not to be a bottleneck for the microprocessor. Figure 1 shows the average IPC for different sizes of the instruction queue and reorder buffer ranging from 16 to 128[1]. We can see that FP programs in general experience a significant benefit for every increase of the instruction window up to 128 entries whereas integer programs reach a plateau for 64 entries. Since we assume an instruction queue that is shared by both integer and FP instructions, we set its size to 128 entries. It is surprising that the *li* benchmark experiences a slight decrease in performance when the number of entries is increased from 8 to 16. This may happen in some cases due to several reasons. For instance, a larger instruction queue may imply more instructions issued from wrong paths. In some other cases, delaying the issue of some instructions may be beneficial (an eager issuing policy is known not to be optimal).

1. For this experiment, we executed only 50 million of instructions per program.

	Applu	Swim	Tomcatv	Wave	Su2cor	Hydro2d	Avg(%)
Inst. Decode	340.9	336.5	351.1	341.9	344.2	349.9	2.8
BTB	143.3	119.7	195.7	149.5	156.4	187.3	1.3
TLB	63.5	53.0	86.7	66.2	69.3	82.9	0.5
L1 I-cache	677.2	565.4	924.5	706.6	738.9	855.0	5.9
L1 D-cache	621.0	518.5	847.8	647.9	677.7	811.6	5.5
L2 cache	1353.9	1130.4	1848.3	1412.6	1477.4	1769.4	12.0
Rename table	1627.9	1672.3	1725.5	1668.8	1724.7	1738.0	13.5
Inst. queue	3124.8	3136.2	3282.9	3160.8	3170.8	3269.5	25.2
ROB	3429.4	3445.7	3394.5	3489.7	3221.5	3348.8	27.1
Int FU	111.6	109.2	110.2	112.3	103.3	108.4	0.9
FP FU	147.7	144.9	145.8	148.6	136.7	143.4	1.2
I/O logic	244.2	203.8	333.4	254.8	266.5	319.1	2.2
Other parts	189.3	180.1	214.9	192.7	200.8	242.3	1.9
Total	12075.8	11615.8	13461.4	12352.6	12288.3	13225.8	100

Table 2: Energy consumption for floating point programs in mJoule

	Perl	Li	M88ksim	Vortex	Compress	Gcc	Avg(%)
Inst. decode	345.9	334.6	335.2	346.6	333.9	349.3	2.8
BTB	164.5	114.9	107.7	169.4	108.8	109.1	1.1
TLB	72.9	50.9	47.7	75.0	48.2	84.2	0.5
L1 I-cache	777.3	542.9	509.0	800.2	514.1	897.9	5.4
L1 D-cache	712.8	497.9	466.8	733.8	471.4	823.4	5.0
L2 cache	1490.0	1040.8	1017.7	1412.6	1477.4	1795.2	11.1
Rename table	1812.1	1879.0	1742.0	1999.8	1027.8	1773.8	13.8
Inst. queue	3351.5	3420.2	3214.9	3645.3	3106.2	2906.2	26.5
ROB	3247.0	3227.3	3315.5	3558.2	3225.7	3499.9	27.1
Int FU	105.2	100.3	104.1	114.3	103.1	109.2	0.9
FP FU	139.2	132.8	137.8	151.3	136.4	144.5	1.1
I/O logic	280.0	195.8	183.5	288.5	185.3	323.8	2.0
Other parts	267.6	227.9	178.1	364.6	403.7	549.9	2.7
Total	12766.8	11765.6	11360.4	13659.7	11142.2	13366.5	100

Table 3: Energy consumption for integer programs in mJoule

4. Energy Breakdown

The energy consumption breakdown for the different blocks of the microprocessor is shown in Tables 2 and 3 for FP and integer programs respectively. The evaluation reveals that a significant part of the energy is spent by the instruction queue and its associated issue logic. This component is responsible for about one fourth of the total energy consumption. The reorder buffer is another important contributor to power consumption, with a share that is similar to the instruction queue. Note that the reorder buffer is used to store results and thus, it is also used to read source operands. This reading and writing activity is responsible for this high energy demands, whereas the energy consumed by the commit activity is much lower. The contribution of the cache memory, including the first level instruction and data cache (L1 Icache and L1 Dcache) and the second level unified cache (unified L2 cache), is close to another fourth of the total. The cache memory represents the largest part of the chip in terms of area. However, its power density and activity factor are smaller than other parts such as the issue logic. In particular, the first level caches have a low miss rate for the majority of applications and thus, very few accesses are performed to the second level. For this

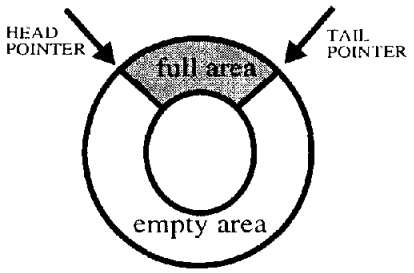


Figure 2: The instruction queue.

reason, the static consumption is a significant component of the energy spent by the second level cache.

Finally, note that the energy breakdown is quite similar for the different programs in spite of their different instruction mix. This is due to the fact that most instructions activate the same blocks of the microarchitecture (e.g. all instructions are fetched, decoded, renamed, issued, and committed using the same circuit blocks). Memory instructions are an exception since they are the only instructions that access the data cache. As we can observe in Table 2 and Table 3, the power consumption of the cache memory shows a significant variation across the different programs.

Increasing the issue width of superscalar processors is an obvious approach to increasing performance (provided that other parts of the processor are scaled accordingly). Most current superscalar processors can issue up to four instructions per cycle but there are already announcements of future microprocessors such as the HAL Sparc64 V [26] that will be able to issue up to eight instructions per cycle. In order to be effective, an increase of the issue width must be accompanied by an increase of the instruction queue size. Some studies suggest that the size of the instruction window must grow more than linearly with respect to the issue width [33]. In addition, the die area of the instruction queue grows more than linearly with respect to the number of entries [35] and the issue width [2]. Thus, we may expect that the relative contribution of the issue logic to the power consumption will grow in future microprocessors.

These results are consistent with some data published for real microprocessors. As an example, the study of Wilcox and Manne [34] about the power consumption of Alpha 21264 shows that the IBox (instruction issue component) has the highest overall power demands and power density of the whole processor and that the issue function represents about 50% of it. They also present the power distribution estimation for the Alpha 21464, where the issue logic will be responsible for the 46% of the total power consumption in the whole chip.

4.1. Detailed Analysis of the Instruction Queue

The previous section showed that the power consumption of the instruction queue represents an important part of the total power consumption and this contribution is likely to augment in future generation microprocessors. The energy consumed by the different entries of the instruction queue is homogeneous. However, we show in this section that the contribution of each entry to the processor performance is very different. The logical structure of the instruction queue is described in Figure 2. We consider the instruction queue implemented as a circular FIFO

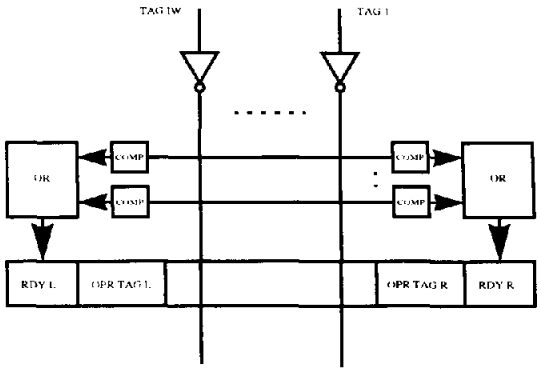


Figure 3: Instruction queue entry.

without collapsing. That is, if an instruction in the middle of the queue is issued, its corresponding entry becomes empty but is not recovered until the head pointer reaches it. Collapsing makes a more effective use of the instruction queue but is much more energy demanding since collapsing implies a shift of all the entries between the tail and the empty entry. A detailed hardware implementation of the instruction queue can be found in [23]. The global structure is designed as a set of CAM cells (see Figure 3). The two basic functions involved in the instruction issue stage are wake-up and select. In addition, the instruction queue is accessed in order to write any new dispatched entry and to read any issued instruction. Among these activities, the wake-up activity is the most energy consuming. On average, for the assumed architecture, it represents 63% of the total power consumption of the instruction queue, which represents 16.3% of the total power consumption of the processor (see Tables 2 and 3). Therefore, in this work we focus on techniques to reduce the dynamic power consumption of the wake-up logic.

Every cycle the wake-up logic broadcasts the result tags through the result buses to all the entries and each entry compares them with their tags to find a match. The conventional approach to build the issue logic is power-inefficient due to the following three observations. The first observation is that the part of the instruction queue from the tail to the head (clockwise) is completely empty and thus does not provide any performance benefit. We refer to this part as the empty area. However, the wake-up logic is also activated for these entries. Second, inside the full area there are empty entries that correspond to instructions already issued but not yet recovered due to the non-collapsing feature but the wake-up logic also operates for them. Finally, note that the entries of the instruction queue can have zero, one or two operands ready. A conventional issue logic keeps trying to wake-up each operand of an instruction even after a match has been found and thus, the operand is ready.

We have quantified the contribution of each of these three sources of energy waste. The first two columns in Table 4 show the average number of entries in the full area of the instruction queue and the average number of empty entries in the full area respectively. The third column presents the energy consumption wasted by both the empty area and the empty entries in the full area. The fourth column shows the energy wasted by trying to wake-up operands that are ready. All percentages are relative to the energy consumption of the wake-up logic.

We note that on average the full area consists of just 58 out of the 128 entries and 26 of these entries are empty. Empty entries

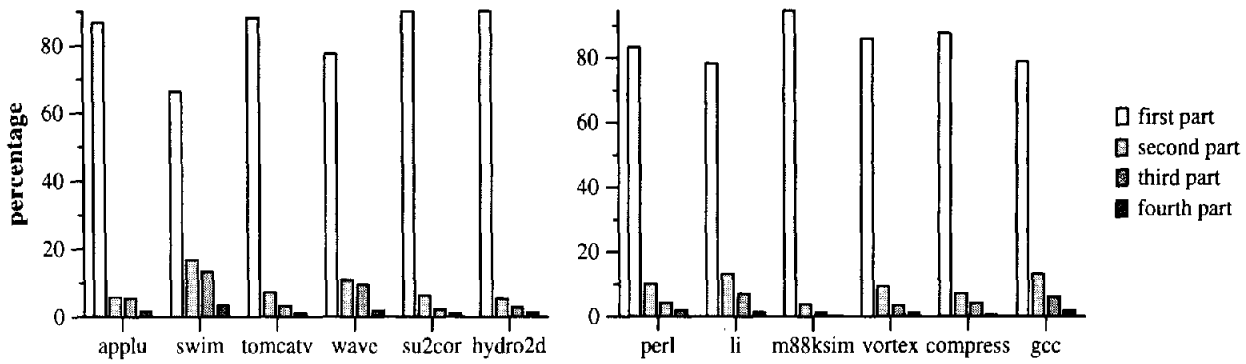


Figure 4: Distribution of useful instructions issued from the four parts of the instruction queue. Each part corresponds to 32 entries.

	Average size of full area (entries)	Empty entries in full area	Energy wasted by empty entries (%)	Energy wasted by ready opnd. (%)	Total energy waste (%)
Applu	93	44	61.8	17.8	79.6
Swim	114	66	62.6	22.2	84.8
Tomcatv	63	29	73.4	14.7	88.1
Wave	103	61	67.2	16.4	83.6
Su2cor	53	19	73.2	15.9	89.1
Hydro2d	49	21	77.6	12.6	90.2
Perl	30	11	84.9	8.8	93.7
Li	25	5	84.3	9.6	93.9
M88ksim	49	9	68.5	18.7	87.3
Vortex	67	27	68.5	18.4	86.9
Compress	25	8	86.8	7.5	94.3
gcc	21	9	90.6	5.5	96.1
Avg	58	26	74.9	14	88.9

Table 4: Statistics for the conventional wake-up logic.

consume 74.9% of the total energy consumed by the wake-up logic. Furthermore, the unnecessary wake-up for ready operands contributes with an additional 14% of energy waste. In total, 88.9% of the energy consumed by the wake-up logic is wasted in these useless activities.

The second observation is that at a given point in time, some parts of the instruction queue may provide a very small benefit in terms of performance even if they are not empty. This is due to the following reasons:

- In periods of the execution with much parallelism, only a subpart of the instruction queue, specially the oldest part, is enough to provide as many instructions as the issue width.
- In periods of execution with little parallelism, some parts of the instruction queue, specially the youngest part, hardly provide any useful instruction ready to be executed.

To provide evidence of this claim we have logically divided the instruction queue in four parts of 32 entries each. The first part is the oldest part and the fourth part is the youngest one. In every portion of the instruction queue we have counted the number of useful (i.e. committed) instructions that are issued along the execution of the program. The results are shown in Figure 4. First, we observe that the number of instructions issued from the youngest part of the instruction queue represent a very low percentage of the total number of useful instructions. This percentage is somewhat higher for the second and third portions. Finally, we can observe that most of the useful instructions are issued from the oldest part of the instruction queue.

In addition, the contribution to IPC (instructions committed per cycle) of a given part of the instruction queue is different for each program, but it varies even for the same program along the time. For instance, Figure 5 shows the number of useful instructions per cycle that are issued from the youngest half part of the instruction queue, for the *applu* program during 15 million of cycles. We can observe that during some periods of time the contribution to IPC of the youngest part of the queue is negligible whereas in some other periods it is as high as one instruction per cycle.

To summarize, a large instruction queue is needed to achieve high performance, but the mechanism devoted to find ready instructions for execution is very power-inefficient because every cycle it consumes a large amount of energy just to check entries that hardly contribute to performance or do not contribute at all. In general, the instructions of the youngest part of the queue are the least useful from a performance standpoint but their usefulness varies for different programs and along the execution of the same program. We can see that there are periods of time when the youngest part of the instruction queue hardly contributes with any useful instruction whereas in other periods its contribution is high.

5. Reducing the Energy Consumption of the Issue Logic

In this section we present a design of the issue logic that significantly reduces its energy consumption with a negligible impact on performance. This design is based on the observations drawn in the previous section.

5.1. Disabling the Wake-up for Empty Entries

The first optimization we propose is to dynamically disable the wake-up function for the empty entries of the instruction queue. These include all entries in the empty area and empty entries in the full area.

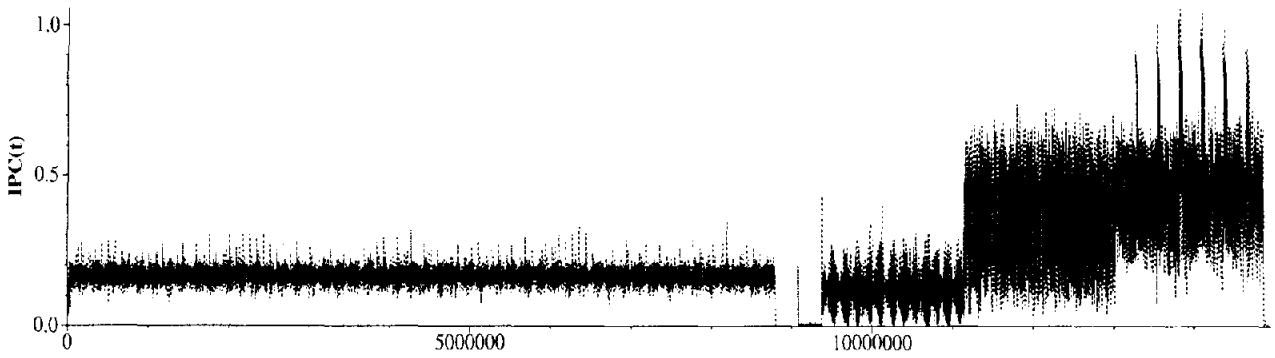


Figure 5: IPC contributed by the youngest part of the instruction queue for applu.

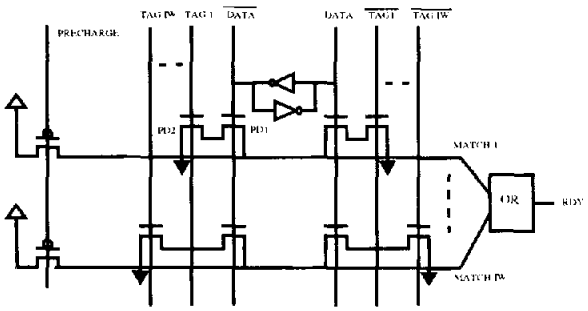


Figure 6: CAM cell wake-up logic [23]

The basic idea is to "gate off" the tag comparisons for all the empty entries. As described above, each instruction queue entry is a CAM cell, which is typically implemented through dynamic logic [37] as shown in Figure 6. Every cycle each match line is precharged and then it is conditionally discharged depending on the values of the tag stored in the cell and the tag forwarded through the corresponding result tag bus. Disabling the precharge will practically save all dynamic energy consumption. Disabling the precharge for empty entries can be achieved adding and AND gate to the precharge signal. The inputs of this gate are the original precharge signal and the valid bit that indicates whether an entry is empty. The output will drive the precharge transistors of every match line. Note that for each operand, there are as many match lines as number of result tag buses. This scheme avoids the energy consumption of all the empty entries, which represent about 74.9% of the total energy consumption of the issue logic as shown in Table 4 and 12.3% of the energy consumed by the whole processor. In fact, the savings will be a bit lower if we take into account the energy consumption of the additional logic, the static energy consumption of the gated CAM cells, and the energy consumed by the tag buses, but these components are relatively low.

5.2. Disabling the Wake-up for Ready Operands

The second proposed optimization allows to disable the wake-up activity for those operands that are already marked as ready. A conventional issue logic keeps on trying to wake-up each ready operand until the instruction is selected for execution, which is obviously a useless activity. Note that in particular, the time since the first operand becomes ready until the instruction is selected can be large.

Disabling this unneeded activity is accomplished again by disabling the precharge operation for all the match lines of ready operands. In this case, the precharge signal is computed as the AND of the original precharge and the bit that indicates whether the operand is ready.

5.3. Dynamic Resizing of the Instruction Queue

In order to exploit the fact that the youngest part of the window provides almost no additional performance for some periods of time, we propose a technique to dynamically resize the instruction queue. The objective is to obtain almost the same IPC as the conventional scheme, but with a lower energy consumption. The basic idea is to implement a run-time mechanism that adapts the size of the instruction queue based on the contribution of the youngest part of the queue to performance. The resizing of the queue is implemented by introducing a limit in the number of entries in the full area, and dynamically adjusting this limit. By reducing the maximum size of the full area, the number of empty entries will increase and the energy will be reduced by the wake-up disabling mechanism described in the previous section.

On the other hand, this resizing should be made in such a way that the performance of the processor is hardly affected by it. In other words, the size of the instruction queue should be reduced when the youngest part hardly contributes to performance and should be increased when there are expectations that the additional part may produce a noticeable improvement of the instruction execution rate.

For reducing the size of the instruction queue, we propose to monitor the contribution of the youngest part of the queue (a chunk of a determined size), and to measure how much these entries contribute to the IPC. If their contribution is negligible, the instruction queue size is reduced.

On the other hand, the size of the instruction queue is increased periodically if it is smaller than its maximum size. Then, it is measured whether this increase provides some benefit in performance, and if not, its size is reduced again to its previous value. Below we detailed the implementation of this mechanism.

The instruction queue is divided into sixteen equal parts of eight entries each. We refer to each part as a portion. In addition to

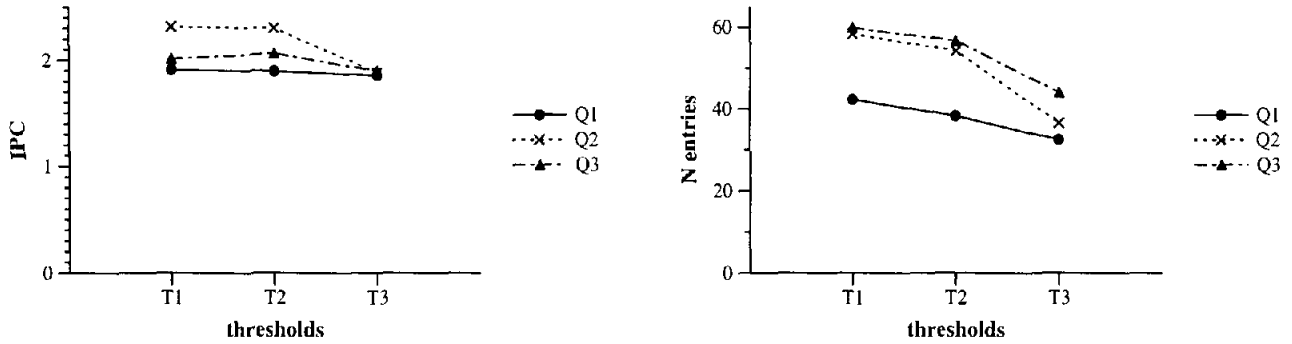


Figure 8: Sensitivity analysis of the quantum and threshold parameters: Q1=100, Q2=1000, Q3=10000 cycles. T1=0.015, T2=0.025, T3=0.05 IPC.

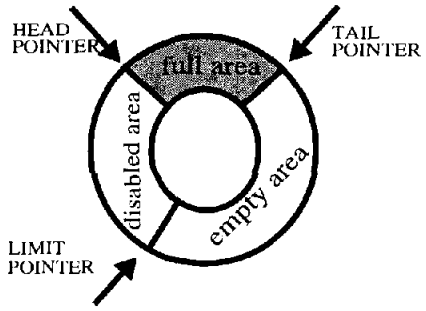


Figure 7: The instruction queue with resizing capabilities.

the usual head and tail pointers, there is an additional pointer that is referred to as the limit pointer (see Figure 7). It determines the furthest point that the tail pointer can reach. In other words, once the tail pointer is equal to the limit pointer, no additional instructions are allowed to be dispatched to the instruction queue. The part of the queue between the head and the tail pointers is called the full area; the part between the tail and the limit pointers is referred to as the empty area; finally, the part of the queue between the limit and the head pointers is called the disabled area. Initially the limit pointer is set to the same value as the head pointer (i.e. the allowed capacity of the instruction queue is set to its maximum value).

During normal conditions, the limit pointer is updated with the same offset used to update the head pointer. When a resize action is performed, an offset corresponding to the size of one portion is added to or subtracted from the limit pointer.

Each entry of the reorder buffer is extended with an additional bit that indicates whether the instruction has been issued from the youngest portion of the instruction queue. When the instruction is dispatched, this bit is set to zero and when the instruction is issued, the bit is set if the instruction is in the current youngest portion of the instruction queue, i.e. the portion just below the limit pointer. Every time an instruction that commits has this bit set, a counter is increased. This counter is examined every certain number of cycles (we refer to this parameter as *quantum*) and if it does not reach a given *threshold*, the instruction queue size is reduced by one portion. Besides, the size of the instruction queue is augmented by an additional portion every five quantums, if it is smaller than the maximum size. The evaluation of committed instructions in the added portion during the next quantum will reveal whether the contribution of the additional portion to IPC is higher than the threshold. If it is not, the instruction queue size will be reduced by one portion at the end of the next quantum.

In order to determine appropriate values for the quantum and threshold, we performed a sensitivity analysis with different values[1]. The main results are depicted in Figure 8. The left-hand side figure shows the IPC and the right-hand side one shows the average number of entries in the full area averaged for the whole benchmark suite. Regarding IPC we can observe that a 1000-cycle quantum outperforms the other two quantums for thresholds 0.015 and 0.025, and achieves a similar result for threshold 0.05. Besides, thresholds 0.015 and 0.025 achieve similar IPCs whereas threshold 0.05 incurs in a significant IPC degradation. On the other hand, decreasing the quantum increases the resizing capability since the mechanism reacts sooner to the varying requirements of the program. We can also observe that the higher the threshold the better the resizing capabilities. Based on these observations, a 1000-cycle quantum and a threshold equal to 0.025 IPC seem a good trade-off between performance and resizing capability. This means that the contribution of the youngest portion of the queue must be greater than 25 instructions every 1000 cycles in order not to be cut-off (note that a single integer counter and comparator is required by the control logic). We have observed that this trend also holds for most individual programs despite of their different features. We may thus expect that the selected parameter values are adequate for a large range of applications. Alternatively, one could develop a technique to dynamically adjust these parameters but this may likely be useful only for few applications.

6. Performance Results

In this section we present a performance study of the proposed mechanism. Table 5 shows some performance statistics for each benchmark as well as the average among them. The first column shows the IPC for the conventional fixed-size instruction queue configuration. The next column shows the IPC of the dynamic resizing scheme. The third column shows the difference in IPC between the fixed-size and the dynamic resizing schemes. The next column shows the difference in IPC in percentage with respect to the IPC of the fixed-size configuration. Finally, the last

1. For this experiment, we run 50 million of instructions per benchmark.

	IPC Fixed-size	Dynamic resizing			
		IPC	IPC loss	IPC loss (%)	Avg. size of full area (entries)
Applu	2.757	2.692	0.065	2.3	50
Swim	2.938	2.852	0.086	2.9	89
Tomcatv	2.353	2.301	0.052	2.2	46
Wave	2.479	2.452	0.027	1.1	74
Su2cor	2.363	2.341	0.022	0.9	49
Hydro2d	2.237	2.204	0.033	1.5	42
Perl	2.042	2.018	0.024	1.2	25
Li	1.810	1.777	0.033	1.8	22
M8kksim	2.839	2.787	0.052	1.8	40
Vortex	2.195	2.159	0.036	1.6	45
Compress	2.118	2.089	0.029	1.4	22
Gcc	1.731	1.699	0.032	1.8	10
Avg	2.321	2.280	0.041	1.7	43

Table 5: Performance statistics

	Empty entries (%)	Ready opnd. (%)	Total w.r.t. wake-up (%)	Total w.r.t. whole processor (%)
Applu	83.5	7.7	91.2	14.9
Swim	74.8	13.3	88.1	15.0
Tomcatv	77	12.8	89.8	13.8
Wave	78.9	9.4	88.3	14.2
Su2cor	73.6	15.9	89.5	14.5
Hydro2d	77.3	13.3	90.6	14.1
Perl	88.4	7.2	95.6	15.8
Li	86	8.6	94.6	17.3
M8kksim	74.2	15.9	90.1	16.1
Vortex	75.7	15.3	91	15.3
Compress	88.2	6.8	95	16.7
Gcc	79.5	6	85.5	11.7
Avg	79.7	11	90.7	14.9

Table 6: Energy savings

column shows the average size of the instruction queue (i.e. the size of the full area).

We can observe that the dynamic resizing scheme reduces the size of the instruction queue by 26% on average (i.e. 58 – see Table 4 – to 43 entries). The size reduction is significant for all programs and goes up to 46% for *applu* and 52% for *gcc*. At the same time, this important reduction on the queue size has a negligible impact on performance. The IPC decreases by less than 3% for every program and about 1.7% on average.

Table 6 shows the wake-up energy consumption savings when the dynamic resizing scheme is used. The first column lists the percentage of energy saving due to empty entries. The second column shows the savings due to ready operands. The next column lists the total energy savings with respect to the energy spent by the wake-up logic, and the last column shows the savings relative to the total energy consumption of the processor. We can observe that the energy consumption of the wake-up logic is drastically reduced. The reduction is as large as 95.6% for individual programs, and on average, it is reduced by 90.7%. This represents an average saving of 14.9% of the total power consumption of the processor. The main sources of energy reduction are empty entries and ready operands. Once these optimizations are applied, the dynamic resizing technique reduces the remaining energy consumption by 16% on average.

7. Summary

In modern superscalar processors, one of the main energy consuming parts is the hardware devoted to issuing instructions out-of-order. This consumption is likely to increase in the future since energy consumption is almost proportional to both the number of entries in the instruction queue and the issue width, and we may expect an increase of these two factors in future microarchitectures if we extrapolate the observed trend in the recent past.

Conventional issue schemes are very power inefficient because they perform much unneeded activity, are too rigid and do not adapt to the varying features of different programs and different parts the same program. We have shown that some parts of the instruction queue have a varying contribution to performance, which ranges from significant to null.

In this work we have proposed a mechanism in order to disable useless activity of the issue logic. In particular, the proposed technique gates-off the wake-up for entries that are empty or operands that are ready. Besides, we have proposed a technique to dynamically resize the instruction queue according to the varying parallelism exhibited by the instruction stream. This reduces the number of valid entries in the queue, and increases the savings provided by gating-off the wake-up for empty entries. The results for a typical superscalar microprocessor show that these mechanisms reduce the energy consumed by the wake-up logic by 90.7%, which represents a 14.9% saving of the total energy consumed by the processor. This is achieved with a minimal impact on performance since IPC is degraded by just 1.7%.

Acknowledgements

This work has been partially supported by the CYCIT grant TIC98-0511 and the IHP program, Access to Research Infrastructure under contract HPRI-1999-CT-00071 between the European Community and CESCA-CEPBA. We would like to thank Ramon Canal and Llorenç Cruz for their help preparing the final manuscript.

8. References

[1] G. Albera, I. Bahar, S. Manne "Power and Performance trade-offs using various caching strategies" *Proceedings of the Int'l Symposium on low-Power Electronics and Design*, 1998

[2] D. Albonesi "Dynamic IPC/Clock Rate Optimization" *Proceedings of the 25th Int'l Symposium on computer Architecture*, 1998

[3] D. Albonesi "The inherent Energy Efficiency of Complexity-Adaptive Processors" *Power Driven Microarchitecture Workshop in conjuction with ISCA-25*, 1998

[4] D. Burger, T. Austin "The SimpleScalar Tool Set" , Version 3.0 Technical Report, University of Wisconsin, Madison 1999

[5] D. Brooks, M. Martonosi "Adaptive Thermal Management for High-Performance Microprocessors", *Workshop on Complexity Effective Design in conjuction with ISCA-27* , 2000

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

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

[8] R. Canal, A. González and J. Smith "Very Low Power Pipelines using Significance Compression" *Proc. of 33rd. Int. Symposium on Microarchitecture (MICRO-33), Monterey, CA (USA), December 10-13, 2000*

[9] T. M. Conte, M. Toburen, M. Reilly "Instruction Scheduling for Low power dissipation in High Performance Microprocessors", *Workshop on Power Driven Microarchitecture in conjuction with ISCA-25*,1998

[10] Intel Corporation "The Intel Architecture Software Developers Manual", 1999

[11] Intel Corporation "Intel StrongArm SA-110 Microprocessor Datasheet", 1999

[12] Mentor Graphics Corporation "QuickPower", 1999

[13] Synopsys Corporation "PowerMill Data Sheet", 1999

[14] Transmeta Corporation "The Technology Behind the Crusoe Processor Whitepaper", 2000

[15] J. Cortadella,T. Lang, E. Mussoll "Reducing the energy of address and Data Buses with the Working-Zone Encoding Technique and its Effect on Multimedia Applications", *Workshop on Power Driven Microarchitecture in conjuction with ISCA-25*, 1998

[16] K. Diefendorf, P. K. Dubey "How Multimedia Workloads Will Change Processor Design", *IEEE Computer Journal*, 30(9), 1997, pages 43-45

[17] D. Folegnani and A. González,"Reducing Power Consumption of the Issue Logic", *Proc. of Workshop on Complexity-Effective Design* held in conjunction with ISCA 2000, Vancouver (Canada), June 10, 2000

[18] R. González, M. Horowitz "Energy Dissipation in General Purpose microprocessors", *IEEE Journal of Solid State Circuits*, 31(9), 1996, pages 1277-1284

[19] M. Johnson, W. Mangione-Smith "The filter cache: An energy efficient memory structure" *Proceedings of the Int'l Symposium on Microarchitecture*, 1997

[20] M. Kamble, K. Ghose "Analytical Energy Dissipation Models for low power Caches", *Proceedings of int'l Symposium on low power Electronics and Design*, 1997

[21] A. Klauser, D. Grunwald, S.Manne "Pipeline gating: Speculation control for energy reduction", *Proceedings of the 25th Int'l Symposium on computer Architecture*, 1998

[22] D. Marculescu, "Profile-Driven Code Execution for Low Power Dissipation" in *Proc. of Int. Symp on Low Power Electronics and Design*, pp. 253-255, 2000

[23] S. Palacharla "Complexity effective superscalar processors", PhD Thesis, University of Winsconsin, Madison,1998

[24] M. Panich "Reducing Instruction Cache Energy Using Gated Wordlines", MS Thesis, MIT, 1999

[25] H. Sánchez "Thermal management system for high performance PowerPC microprocessors". *Digest of Technical Papers COMPCON IEEE Computer Society International Conference*, 1997

[26] M Shebanow "SPARC64 5: A High Performance and High Reliability 64-bit SPARC Processor", CSLI Public Event, Stanford University, December 1999

[27] D. Singh, V. Tiwari "Power Challenges in the Internet World", *Proceedings of the CoolChips tutorial, An Industrial Perspective on Low Power Processor Design, in conjunction with MICRO-32*, 1999

[28] C. Small "Shrinking Devices Put the Squeeze on System Packaging", *EDN*, 39(4), 1994, pp 41-46

[29] C. Su, A. Despain "Cache Designs for Energy Efficiency", *Proceedings of the 28th Hawaii Int'l Conference on System Science*, 1995

[30] D. Tennenhouse "Pro-Active Computing", Darpa Technical report, 1999

[31] The CPU Info Center. *http://infopad.eecs.berkeley.edu/CIC*

[32] N. Vijaykrishnan,M. Kandermir,M.J. Irwin, H. Kim,W. Ye "A Unified Energy Estimation Framework with Integrated Hardware-Software Optimizations", *Proceedings of the 27th Int'l Symposium on computer Architecture*, 2000

[33] D. Wall "Limits of Instruction-Level Parallelism", Technical report WRL 93/6, Digital WRL, 1993

[34] K. Wilcox, S. Manne "Alpha Processors: A History of Power Issues and A Look to the Future", *Proceedings of the CoolChips tutorial, An Industrial Perspective on Low Power Processor Design in conjunction MICRO-33*, 1999

[35] V. Zyuban "Inherently Lower-Power High performance Superscalar Architectures", PhD Thesis, University of Notre Dame, 2000

[36] N. Vijaykrishnan, M. Kandermir, M.J. Irwin, H. Kim and W. Ye "Energy-Driven Hardware-Software Optimizations Using SimplePower" *Proc. of the 27th Int'l Symposium on Computer Architecture*, pp. 95-106 June 2000

[37] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Desing*. Addison Wesley, second ed., 1993