Instruction Packing: Reducing Power and Delay of the Dynamic Scheduling Logic

Joseph J. Sharkey, Dmitry V. Ponomarev, Kanad Ghose

Department of Computer Science
State University of New York
Binghamton, NY 13902-6000

{jsharke, dima, ghose}@cs.binghamton.edu

Oguz Ergin*

Intel Barcelona Research Center
Intel Labs, UPC
Barcelona, Spain

oguzx.ergin@intel.com

ABSTRACT

The instruction scheduling logic used in modern superscalar microprocessors often relies on associative searching of the issue queue entries to dynamically wakeup instructions for the execution. Traditional designs use one issue queue entry for each instruction, regardless of the actual number of operands actively used in the wakeup process. In this paper we propose *Instruction Packing* – a novel microarchitectural technique that reduces both the delay and the power consumption of the issue queue by sharing the associative part of an issue queue entry between two instructions, each with at most one non-ready register source operand at the time of dispatch. Our results show that Instruction Packing provides a 39% reduction of the whole issue queue power and 21.6% reduction in the wakeup delay with as little as 0.4% IPC degradation on the average across the simulated SPEC benchmarks.

Categories and Subject Descriptors

C.1.1 [**Computer Systems Organization**]: Processor Architecture – *Pipeline processors*

General Terms: Design, Performance, Measurement

Keywords: Issue Queue, Low Power, Instruction Packing

1. INTRODUCTION

Contemporary superscalar microarchitectures rely on aggressive out-of-order instruction scheduling mechanisms to maximize performance across a wide variety of applications. Instructions are dispatched into the issue queue (IQ) in-order after undergoing register renaming and noting the availability of their register sources, and they are issued for the execution out of order as their source operands become available. The instruction scheduling logic operates in two phases – instruction wakeup and instruction selection. During wakeup, the destination tags of the already scheduled instructions are broadcasted across the IQ and each IQ entry associatively compares the locally stored source register tags against the broadcasted destination tag. On a match, the corresponding source is marked as ready. When all sources of an instruction become ready, the instruction wakes up and becomes eligible for selection. The selection logic then selects W out of possibly N eligible instructions, where N is the size of the IQ and W is the processor's issue width.

ISLPED'05, August 8–10, 2005, San Diego, California, USA.

In order to uncover sufficient instruction-level parallelism, the IQs have to be generously sized, resulting in significant power/energy dissipations in the course of accessing the queue. A large amount of energy is dissipated when the destination tags are broadcasted on the tag busses due to the charging and discharging of the wire capacitance of the tag line itself and the gate capacitance of the devices that implement the tag comparators. Additionally, the power is also dissipated in the course of instruction dispatching (writing the instructions into the IQ) instruction selection, and instruction issuing (reading the instructions out of the queue). Several researchers have reported the IQ power to account for 20%-25% of total chip power [17, 35].

In a traditional RISC-like processor where each instruction can have at most two register source operands, each IQ entry has two comparators, which allow the instruction to track the arrival of both sources by monitoring the tag buses. In general, however, such a design results in a grossly inefficient usage of the comparators, because of two reasons: 1) Many instructions have only one source register operand, and therefore do not require the use of two tags (and two comparators) in the first place, and 2) of the instructions with two register source operands, a large percentage have at least one of these operands ready at the time of dispatch, again rendering the second comparator unnecessary. Our simulations showed that, on the average across SPEC 2000 benchmarks, about 83% of the dynamic instructions enter the scheduling window with at least one of their source operands ready, as shown in Figure 1.

In this paper, we propose to optimize the use of the CAM logic (comparators) within the IQ by the opportunistic packing of two instructions into the same IQ entry, effectively duplicating some of the RAM storage for these instructions (destination register addresses, literals, opcodes) and sharing the existing comparators. When an instruction, which has at least one of its two source registers ready at the time of dispatch, is placed in a traditional IQ entry with two comparators, one of these comparators (the one corresponding to the ready source) is not used. By placing two such instructions within the same IQ entry, such wastages can be reduced and in many cases totally eliminated. Instruction packing reduces the number of IQ entries and thus the length of and the capacitive loading on the tag buses and the bitlines. Consequently, significant energy reduction and faster operation of the wakeup logic are achieved (as the queue now has a significantly smaller number of entries) with negligible degradation in the IPC and just a slight increase in the delay of the selection logic.

---

* This work was done while Oguz Ergin was at SUNY Binghamton.

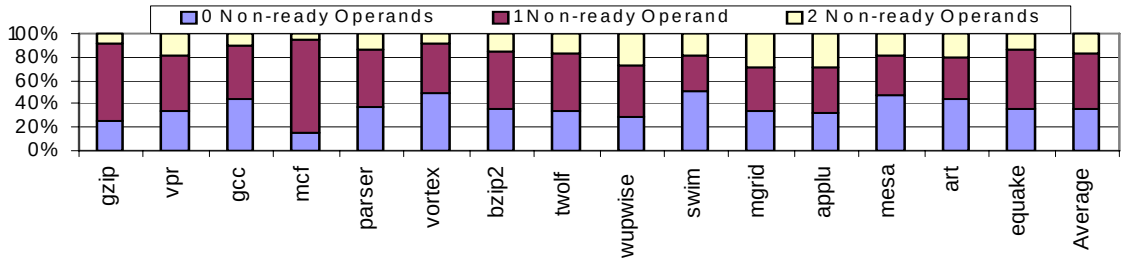


Figure 1: The percentage of dynamic instructions that enter the scheduling window with 0, 1, and 2 operands requiring comparators.

Other researchers [13] previously proposed to *statically* partition the IQ into the entries with 0, 1 and 2 comparators, exploiting the same observation that only a relatively small percentage of instructions come into the pipeline with both of their source registers non-ready. The key difference between the work of [13] and this paper is that instruction packing *dynamically* allocates either a full-entry or a half-entry of the queue to an instruction, and thus it can efficiently adapt to the characteristics of executing applications and encounter lower IPC degradation, as we demonstrate in the results section.

We perform detailed microarchitectural and circuit-level simulations of the proposed mechanism. Our results show more than 39% reduction in the overall IQ power (excluding the selection logic) and 21.6% reduction in the wakeup delay at the cost of IPC degradation of as little as 0.4% on the average across the benchmarks.

The rest of the paper is organized as follows. Section 2 describes instruction packing. Our simulation methodology is described in Section 3 and our simulation results are presented in Section 4. We describe the related work in Section 5 and offer our concluding remarks in Section 6.

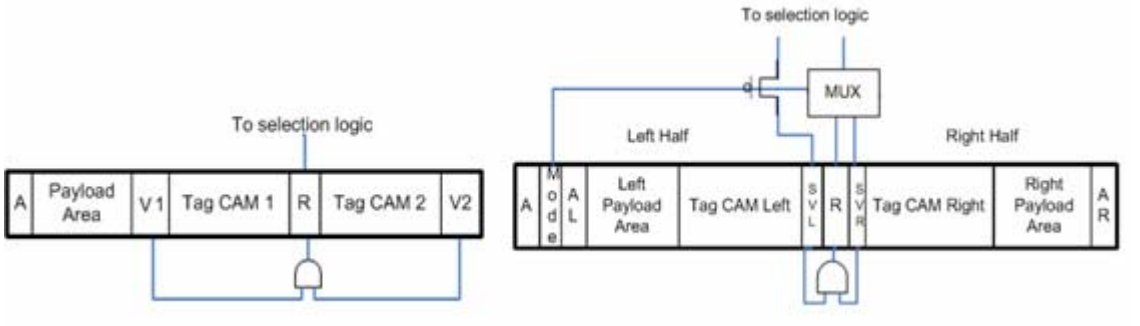
2. INSTRUCTION PACKING

Figure 2(a) shows a format of an IQ entry used in traditional processors. The following fields comprise a single IQ entry: a) entry allocated bit (A), b) payload area (opcode, FU type, destination register address, literals), c) tag of the first source, associated comparator (tag CAM word 1, hereafter just tag CAM 1, without the "word") and the source valid bit, d) tag of the second source, associated comparator (tag CAM 2) and source valid bit, and e) the ready bit. The ready bit, used to raise the request signal for the selection logic is set by AND-ing the valid bits of the two sources.

If at least one of the source operands is ready at the time of dispatch, the tag CAM associated with this instruction's IQ entry remains unused. To exploit this idle tag CAM, we propose to share one IQ entry between two such instructions. An entry in the IQ can now hold one or two instructions, depending on the number of ready operands in the stored instructions at the time of dispatching. Specifically, if both source registers of an instruction are not available at the time of dispatch, the instruction is assigned an IQ entry of its own and makes use of both tag CAMs in the assigned entry to determine when its operands are ready. An instruction that has only one source register that is not available at the time of dispatch is assigned just one half of an IQ entry. The remaining half of the IQ entry may be used by another instruction that also has one of its source registers unavailable at the time of dispatch. Sharing an IQ entry between two instructions also requires the IQ entry to be widened to permit the payload parts of both instructions to be stored, along with the addition of flags that indicate whether the entry is shared between two instructions and the status of the stored instruction(s). Figure 2(b) shows the format of an IQ entry that supports instruction packing. Each IQ entry is comprised of the "entry allocated" bit (A), the ready bit (R), the mode bit (MODE) and the two symmetrical halves: the left half and the right half. The structure of each half is identical, so we will use the left half for the subsequent explanations.

A left half of each IQ entry contains the following fields:

- Left half allocated (AL) bit. This bit is set when the left half-entry is allocated.
- Source tag and associated comparator (Tag CAM). This is where the tag of the non-ready source operand for an instruction with at most one non-ready source is stored.
- Source valid left bit (SVL). This bit signifies the validity of the source from part b), similar to the traditional designs. This bit is also used to indicate if the instruction residing in a half-entry is ready for selection (as explained later)
- Payload area. The payload area contains the same information as in the traditional design, namely: opcode, bits identifying the FU type, destination register address and literal



(a) **(b)**

Figure 2: Formats of a traditional issue queue entry (left), and an entry of the issue queue that supports instruction packing

bits. In addition, the payload area contains the tag of the second register source. Notice that the tag of the second source does not participate in the wakeup, because if an instruction is allocated to a half-entry, the second source must be valid at the time of dispatch. Compared to the traditional design, the payload area is increased by the number of bits used to represent a source tag.

The contents of the right half are similar. The ready bit (R) is used when an instruction with two non-ready source operands is allocated into the full IQ entry, as explained below. In summary, each entry in the modified IQ is divided into a left half and a right half, each is capable of storing an instruction with at most one non-ready source operand, or the two halves can be used in concert to house an instruction with two non-ready source operands.

In general, the IQ entry can be in one of the following three states: 1) the entry holds a single instruction, both register source operands of which were not ready at the time of dispatch, 2) the entry holds two (or one with another half free) instructions, each of which had at least one source operand ready at the time of dispatch, or 3) the entry is free. The "mode" bit, stored within each IQ entry as shown in Figure 2(b), identifies the state of the entry. If the mode bit is set to 1, then the entry maintains a two-operand instruction, otherwise it either maintains one or two single-operand instructions or it is free.

Since each entry can hold up to two instructions, fewer IQ entries are needed. However, despite the fact that each entry in the modified IQ shown in Figure 2(b) is somewhat wider than the traditional queue entry (due to the replication of the Payload area and three extra bits – AL, AR, and MODE), the amount of CAM logic per-entry does not change. Each entry still uses only two sets of comparators – those are either used by one instruction that occupies the full entry, or are shared by two instructions, each located in a half-entry. In the next few subsections, we describe the details of this technique.

2.1 Entry Allocation

To set up an IQ entry for an instruction, the "*entry allocated*" bits corresponding to both halves (AL and AR), as well as the global *entry_allocated* bit (A) are associatively searched in parallel with register renaming and checking the status of the source physical registers. If the instruction is determined to have at most one non-ready source operand at the time of dispatch, the lowest numbered IQ entry with at least one available half is allocated. If both halves are available within the chosen entry, then the instruction is written into the right half. After the appropriate half is chosen, both the "*entry_allocated*" bit of this half and the global A bit are reset. If an instruction is determined to have two non-ready source operands at dispatch, then a full-sized entry is allocated, as dictated by the state of the A bits. The search for a full-sized and a half-sized entry occurs simultaneously, and the entry to be allocated is then chosen based on the number of non-ready source operands. This IQ entry allocation process is somewhat more complicated than similar allocation used in traditional designs, where just the A bits are associatively searched. However, there is no extra delay involved, because the searches occur in parallel. Similar issues with allocating the IQ entries are also inherent in other designs which aim to reduce the amount of associative logic in the queue by placing the instructions into the IQ entries judiciously, based on the number of non-ready operands at the time of dispatch [13]. We will discuss what kind of information is written into the IQ for the various instruction categories later in the paper. But first, we describe how wakeup and selection operations are implemented in this scheme.

2.2 Instruction Wakeup

The process of instruction wakeup remains exactly the same as in traditional design for an instruction that occupies a full IQ entry (i.e. enters the queue with two non-ready register sources). Here, the ready bit (R) is set by AND-ing the valid bits of both sources. For instructions that occupy half of an IQ entry, the wakeup simply amounts to setting of the valid bit corresponding to the source that was non-ready when the instruction entered the IQ. The contents of the source valid bits are then directly used to indicate that the instruction is ready for selection (the validity of the second source is implicit in this case). The selection logic details are described next.

2.3 Instruction Selection

The process of instruction selection needs to be slightly modified to support instruction packing. To make the explanation easier, we assume that a 32-entry IQ is packed into a 16-entry structure, such that each entry is capable of holding two instructions with at most one non-ready source each, or one instruction with two non-ready sources. In a 32-entry IQ design, there are 32 request lines that can be raised by the awakened instructions – one line per IQ entry. In the instruction packing scheme, each of the two halves of each of the 16 entries requires a request line, thus retaining the same total number of request lines (32) and resulting in a similar complexity of the selection logic. In addition, the ready bits, used by the instructions allocated to full entries, also require request lines. Consequently, a straightforward implementation of the selection logic would require 48 (3x16) request lines, thus increasing the complexity, delay and power requirements of the select mechanism.

Such an undesirable elevation in the complexity of the selection logic can be avoided by sharing one request line between the R and the SVR bits. The shared request line is raised if at least one of the bits (the R or the SVR) is set. The R and the SVR bits are both connected to the shared request line through a multiplexer, which is controlled by the "mode" bit of the IQ entry (Figure 2(b)).

Consequently, the overall delay of the selection logic increases only slightly – by the delay of a multiplexer. Notice also that the MUX control signal (the "mode" bit) is available in the beginning of the cycle when the selection process takes place (the "mode" bit is set when the IQ entry is allocated). The request line driven by the SVL bit is controlled by the p-device, whose gate is connected to the "mode" bit. This request line will be asserted only if the "mode" bit is set to 0 (indicating that the IQ entry is shared between two instructions) and the SVL bit is set to 1.

Note that the only part of the selection logic that is modified is the process of asserting the request lines. The rest of the selection logic is unchanged compared to the traditional designs. The overall delay of the selection logic is thus increased by the delay of the multiplexer, whose control signal is preset (as the value of the "mode" bit is available as soon as the IQ entry is allocated). The additional delay introduced in path that generates the request signal is thus the propagation delay of a turned-on CMOS switch within the multiplexer (about 18 ps in our 0.18 micron CMOS layouts). Compared to the delay of a 3-level, tree structured selection logic (about 370 ps, *neglecting wire delays*), this additional delay is negligible, only about 5%. This percentage, of course, goes down as wire delays in the selection tree are accounted for.

2.4 Instruction Issue

We define instruction issue as a process of reading the source operand tags of the selected instructions and starting the register file access (effectively moving the instruction out of the IQ). When a grant signal comes back corresponding to the request line, which was shared between the R and the SVR, the issue logic has to know which physical registers have to be read. Conventionally,

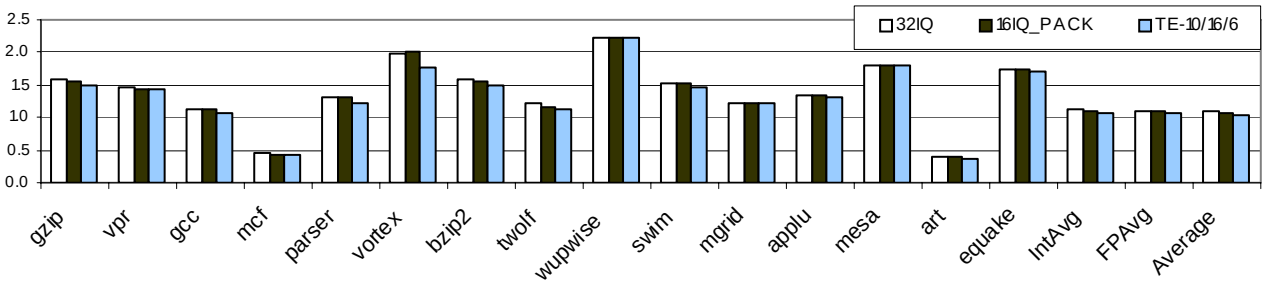


Figure 3: Commit IPCs for machines with various scheduler designs

this information is conveyed by the contents of the tag fields. However, the register tags of an instruction with two non-ready sources (i.e. the instruction that occupies full IQ entry) and the register tags of an instruction with one non-ready source are generally stored in different locations within the IQ entry. In the former case, the tags are stored in the tag fields connected to both comparators – one tag is stored in the left half of the entry and the other tag is stored in the right half of the entry. In the latter case, both tags are stored in the right half of the entry, such that the tag of the non-ready operand is connected to the comparator and the other tag is simply stored in the payload area. Given this disparate locations of the source register tags, how would the issue logic know which tags to use when the grant signal corresponding to a shared request line comes back?

One solution is, again, to use the contents of the "mode" bit and a few multiplexors. This will, however, slightly increase the delay of the issue/register access cycle. A better solution, which avoids the additional delays in instruction issuing altogether, is as follows. When an instruction with two non-ready sources is allocated to the IQ, the tag, which is connected to the left half comparator, is also replicated in the payload area storage for the second tag in the right half. As a result, both tags will be present in the right half of the queue, so these tags can be simply used for register file access, without regard for the IQ entry mode.

2.5 Benefits of Instruction Packing

Instruction packing, as described in this section, has several benefits over the traditional IQ designs in terms of layout area, access delays and power consumption.

The delay of the wakeup logic is reduced, because the tag buses become much shorter and the capacitive loading on these buses is also significantly reduced – the delay in driving the tag bus (which is a major component of the wakeup latency) is roughly reduced by half. Furthermore, shorter bitlines reduce the IQ access delays during instruction dispatching (setting up the entries) and issuing (reading out the register tags and literals). Finally, for the same reasons the power consumptions of instruction dispatch, issue, and wakeup are also reduced. Another potential reason for the reduction in the power consumption has to do with the use of fewer comparators. In the instruction packing, the tags of the source registers ready during dispatching are never associated with the comparators. In the traditional designs, each and every source tag is hooked up to a comparator. Unless these comparators are precharged selectively (based on whether or not a given IQ slot is awaiting for the result), unnecessary dissipations can occur when comparators associated with the already valid sources discharge on subsequent mismatches.

Finally, with instruction packing, the area of the IQ decreases, because compared to the traditional designs, the amount of RAM storage does not change (we use twice as fewer entries, but each entry has about twice the amount of RAM), but the amount of associative logic is reduced by a factor of two.

In the results section, we quantify these savings using detailed simulations of SPEC 2000 benchmarks and also circuit simulations of the IQ layouts. Notice that all these benefits are achieved with essentially no degradation in the IPCs (committed Instructions Per Cycle). This is because most instructions (our results show 83%) have at least one of their sources ready at the time of dispatch, thus rendering the performance loss due to the smaller number of IQ entries negligible.

3. SIMULATION METHODOLOGY

Our simulation environment includes a detailed cycle-accurate simulator of the microarchitecture and cache hierarchy. We used a modified version of the Simplescalar simulator [6] that implements separate structures for the IQ, re-order buffer, load-store queue, register files, and the rename tables in order to more accurately model the operation of modern processors. All benchmarks were compiled with gcc 2.6.3 (compiler options: -O2) and linked with glibc 1.09, compiled with the same options, to generate the code in the portable ISA (PISA) format. All simulations were run on a subset of the SPEC 2000 benchmarks consisting of 8 integer and 7 floating-point benchmarks using their reference inputs. In all cases, predictors and caches were warmed up for 1 billion committed instructions and statistics were gathered for the next 500 million instructions. Table 1 presents the configuration of the baseline simulated processor.

For estimating the delay, energy and area requirements, we designed the actual VLSI layouts of the IQ and simulated them using SPICE. The layouts were created in a 0.18 micron 6 metal layer CMOS process (TSMC) using Cadence design tools. A Vdd of 1.8 volts was assumed for all the measurements.

Table 1. Configuration of the Simulated Processor

Parameter	*Configuration*
Machine width	4-wide fetch, 4-wide issue, 4 wide commit
Window size	issue queue – as specified,48 entry load/store queue, 96–entry ROB
Function Units and Latency (total/issue)	4 Int Add (1/1), 2 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 2 FP Add (2), 2 FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
Phys. Registers	128 combined integer + floating-point
L1 I–cache	64 KB, 1–way set–associative, 128 byte line, 1 cycles hit time
L1 D–cache	64 KB, 4–way set–associative, 64 byte line, 2 cycles hit time
L2 Cache unified	2 MB, 8–way set–associative, 128 byte line, 6 cycles hit time
BTB	2048 entry, 2–way set–associative
Branch Predictor	Combined with 1K entry Gshare, 10 bit global history, 4K entry bimodal, 1K entry selector
Branch Mispred. Pen.	8 cycles minimum
Memory	128 bit wide, 150 cycles first chunk, 1 cycles interchunk
TLB	32 entry (I), 128 entry (D), fully associative

4. RESULTS

This section presents the performance, power, and delay analysis of instruction packing. Figure 3 shows the commit IPCs for the microarchitectures using three different scheduling schemes: the baseline scheduler (the leftmost bar), instruction packing (the middle bar) and the scheduler that uses tag

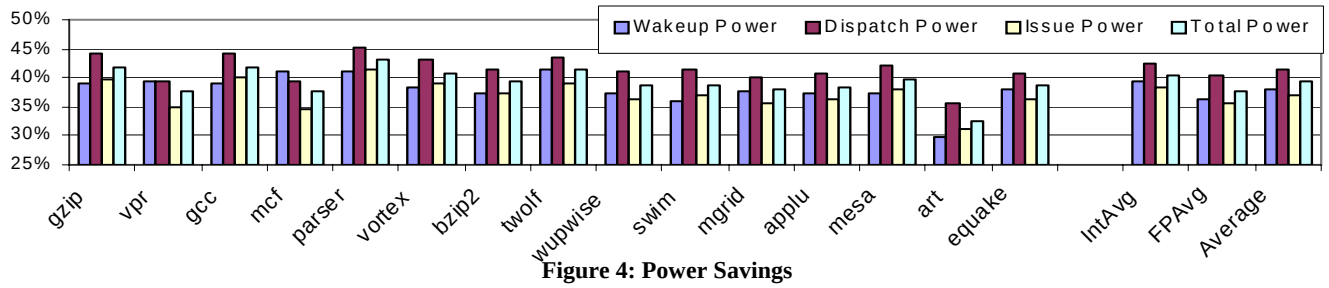


Figure 4: Power Savings

elimination (TE) design of [13] (rightmost bar), which statically partitions the IQ providing some entries with 2 comparators, others with 1 comparator and yet others with 0 comparators. The TE bar is only presented for comparison purposes - detailed qualitative comparison between instruction packing and the TE scheme is given in Section 5. The baseline scheduler uses a 32-entry IQ, the scheduler implementing instruction packing uses 16-entry IQ, and the TE design uses the configuration 10/16/6 (10 2-comparator entries, 16 single-comparator entries and 6 entries with no comparators). This corresponds to the best configuration presented in [13]. Also, according to our results, the distribution of operands with 0, 1 or 2 non-ready sources roughly corresponds to the TE configuration presented in Figure 3. The 16-entry scheduler with instruction packing performs within 0.4% of a traditional 32-entry IQ, on average, with the worst case being 4.0% degradation in *twolf* and less than 0.01% performance degradation in 12 of the benchmarks (*gzip*, *gcc*, *mcf*, *parser*, *vortex*, *wupwise*, *swim*, *mgrid*, *applu*, *mesa*, *art*, *equake*). This result is expected because most instructions fit in a half-entry in the queue utilizing packing, effectively doubling its capacity. The TE scheduler has 2.6% IPC degradation on the average, similar result was also presented in [13]. Consequently, the instruction packing almost completely eliminates the IPC loss incurred by the TE. This is a result of dynamically allocating the IQ entries as opposed to static partitioning of the queue used in the TE scheme.

CMOS layouts of both the 32-entry traditional queue and the 16-entry packing queue show a 26.7% reduction in the IQ area due to the use of instruction packing. Packing effectively reduces the number of CAM bitcells by half, while increasing the number of SRAM bitcells in each row (but leaving the total number of SRAM bitcells in the IQ practically unchanged). As observed in our layouts, CAM bitcells are 41% larger than their SRAM counterparts. It is the reduction in the number of CAM bitcells that accounts for the IQ area reduction.

Table 2: Wakeup delays of a 16-entry queue supporting instruction packing compared to a 32-entry traditional queue

	Tag-Bus Drive (ps)	Comparator Output (ps)	Final Match Signal (ps)	Total Delay (ps)
32-entry	224	219	126	569
16-entry Packing	131	201	114	446
Savings:	41.5%	8.2%	9.5%	**21.6%**

As presented in Table 2, instruction packing achieves a 21.6% reduction in the wakeup delay. This delay reduction comes mainly from the shorter and lower-capacitance tag busses and bitlines. Combined with the 5% increase in the delay of the selection logic (as conservatively estimated at the end of Section 2.3) this results in an overall reduction of 11% in the overall scheduler (wakeup/selection) delay.

Finally, the instruction packing saves energy due to the smaller number of tag comparators and shorter tag-busses and bitlines. Figure 4 presents the power savings of the 16-entry packing queue, as extracted from the SPICE simulations. The bars represent the savings in the wakeup power, dispatch power, issue power and the total IQ power. The power of the selection logic was not computed, but it generally represents a small percentage of the overall IQ power and does not change with the instruction packing. Instruction packing saves 39% of total IQ power as compared to a traditional 32-entry queue.

Note that in all power measurements, the dissipation of any additional logic is accounted for when computing the power savings.

5. RELATED WORK

Researchers have proposed several ways to reduce the power consumption of the issue logic. Dynamic adaptation techniques [7,8,16,28] partition the queue into multiple segments and deactivate some segments periodically, when the applications do not require the full IQ to sustain the commit IPCs. Energy-efficient comparators, which dissipate energy predominantly on a tag match were proposed in [26,27]. Also in [27], the IQ power was reduced by using zero-byte encoding and bitline segmentation. In [19], the associative broadcast is replaced with indexing to only enable a single instruction to wakeup. This exploits the observation that many instructions have only one consumer. In [2], the wakeup width is decreased to be smaller than the machine width, noting that the full machine width is rarely used for instruction wakeup.

The observation that many instructions are dispatched with at least one of their source operands ready is not new – it was used in [13], where the scheduler design with reduced number of comparators was proposed. In that scheme, some IQ entries have two comparators, others have just one comparator, and yet others have zero comparators. Despite significant reduction in the number of comparators, the size of the IQ was not reduced. Further, the IPC losses coming from statically partitioning the queue are significantly higher than in the instruction packing, which dynamically adapts the queue to any particular instruction mix. We presented a detailed performance comparison with the scheme of [13] (without last-tag speculation mechanism) in Figure 3 and discussed the results in Section 4. The last-tag speculation mechanism introduced in [13] can further reduce the number of required comparators, but it requires the extra logic to maintain the predictions and also handle possible mispredictions. Some issues with integrating the design of [13] with speculative schedulers are discussed in [20]. In contrast, instruction packing is a completely deterministic scheme, just like the baseline model.

In [20], the tag buses were categorized into fast buses and slow buses, such that the tag broadcast on the slow bus takes one additional cycle. The design again relied on the last-arriving operand prediction to hook the last arriving operand (which actually identifies when the instruction wakes up) to the fast bus to avoid the wakeup delays. The approach [20] does not reduce or share the number of IQ entries and the length of tag buses – it just decouples half of the tag buses from the fast wakeup bus. While we did not perform direct comparison with the work of [20] in terms of IPCs, according to the results presented in their paper, the average IPC loss was 2.2%, which is higher than what is

achieved with instruction packing. Also, power reduction was not the goal of the work of [20].

One approach to reducing scheduling complexity involves pipelining the scheduling logic into separate wakeup and select cycles [21,34]. [34] uses the status of an instruction's grandparents to wakeup the instruction earlier in a speculative manner. [21] proposed grouping of two (or more) dependent single-cycle operations into so-called Macro-OP (MOP), which represents an atomic scheduling entity with multi-cycle execution latency. The concept of dataflow mini-graphs [3] is similar to Macro-Op scheduling in that groups of instructions are scheduled together.

Other proposals have introduced new scheduling techniques with the goal of designing scalable dynamic schedulers [4,12,23,29,33,31]. Brown et.al. [5] proposed to remove the selection logic from the critical path by exploiting the fact that the number of ready instructions in a given cycle is typically smaller than the processor's issue width.

Scheduling techniques based on predicting the issue cycle of an instruction [1,9,10,14,18,22,24] remove the wakeup delay from the critical path and remove the CAM logic from instruction wakeup, but need to keep track of the cycle when each physical register will become ready. In [16], the wakeup time prediction occurs in parallel with the instruction fetching. In [30], a wakeup-free scheduler without counting and issue time estimation logic is proposed.

6. CONCLUDING REMARKS

We proposed Instruction Packing – a novel microarchitectural technique that reduces both the delay and the power consumption of the IQ by sharing the associative part of an IQ entry between two instructions, each with at most one non-ready source. Consequently, the number of IQ entries, and thus the length of and the capacitive loading on the tag busses and bitlines, can be reduced substantially, leading to faster access and lower power dissipation. In addition, the layout area of the IQ is also reduced.

The use of instruction packing results in a 39% reduction in the total IQ power. Additionally, the wakeup delay is reduced by 16%. Combined with the 5% increase in the delay of the selection logic, this results in an overall reduction of 11% in the overall scheduler (wakeup/selection) delay with only 0.4% IPC degradation on the average across SPEC 2000 benchmarks.

7. REFERENCES

[1] J. Abella, A.Gonzalez, "Low-Complexity Distributed Issue Queue", in Proc. Intl. Conf. on High Perf. Computer Architecture, 2004.

[2] A. Aggarwal, et. al., "Defining Wakeup Width for Efficient Dynamic Scheduling", in Proc. of Intl. Conf. on Computer Design, 2004.

[3] A. Bracy, et. al. "Dataflow Mini-Graphs: Amplifying Superscalar Capacity and Bandwidth", in Proc. Intl. Symposium on Microarchtiecture, 2004.

[4] E. Brekelbaum et. al., "Hierarchical Scheduling Windows", in Proc. Intl. Symposium on Microarchitecture 2002.

[5] M. Brown, J. Stark, Y. Patt. "Select-Free Instruction Scheduling Logic", in Proc. Intl. Symposium on Microarchitecture, 2001.

[6] D. Burger and T. Austin, "The SimpleScalar tool set: Version 2.0", Tech. Report, Dept. of CS, Univ. of Wisconsin-Madison, June 1997 and documentation for all Simplescalar releases.

[7] A. Buyuktosunoglu, et.al.,"A Circuit-Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors", in Proc of Great Lakes Symposium on VLSI Design, 2001.

[8] A. Buyuktosunoglu et.al., "Energy-Efficient Co-adaptive Instruction Fetch and Issue", in Proc. Intl. Symposium on Computer Architecture, 2003.

[9] R.Canal, A. Gonzalez, "A Low-Complexity Issue Logic", in Proc. Intl. Conference on Supercomputing, 2000.

[10] R.Canal, A.Gonzalez, "Reducing the Complexity of the Issue Logic", in Proc. Intl. Conference on Supercomputing, 2001.

[11] Z. Chishti, T. Vijaykumar, "Wire Delay Is Not a Problem for SMT", in Proc. Intl. Symp. on Computer Architecture 2004.

[12] A. Cristal, et.al., "Out-of-Order Commit Processors", in Proc. Intl. Symposium on High Performance Computer Architecture, 2004.

[13] D. Ernst, T. Austin, "Efficient Dynamic Scheduling Through Tag Elimination", in Proc. Intl. Symp. on Computer Architecture, 2002.

[14] D. Ernst, A. Hamel, T.Austin, "Cyclone: a Broadcast-free Dynamic Instruction Scheduler with Selective Replay", in Proc. Intl. Symposium On Computer Architecture, 2003.

[15] T. Ehrhart, S. Patel, "Reducing the Scheduling Critical Cycle using Wakeup Prediction", in Proc. of Intl. Conf. on High Perf. Computer Architecture, 2004.

[16] D.Folegnani, A.Gonzalez, "Energy-Effective Issue Logic", in Proc of Intl. Symposium on Computer Architecture 2001.

[17] M.K. Gowan, L.L. Biro, D.B. Jackson, "Power considerations in the Design of the Alpha 21264 microprocessor", in the Proceedings of the 35th ACM/IEEE Design Automation Conf. (DAC 98), 1998.

[18] J. Hu, N. Vijaykrishnan, M. Irwin, "Exploring Wakeup-Free Instruction Scheduling", in Proc. Intl. Symposium on High Performance Computer Architecture, 2004.

[19] M.Huang et.al., "Energy-Efficient Hybrid Wakeup Logic", in Proc of Intl. Symposium on Low-Power Electronics and Design, 2002.

[20] I.Kim, M.Lipasti, "Half-Price Architecture", in Proc of Intl. Symposium on Computer Architecture, 2003.

[21] I. Kim and M. Lipasti, "Macro-Op Scheduling: Relaxing Scheduling Loop Constraints", in Proc. Intl. Symp. on Microarchitecture, 2003.

[22] Y. Liu, et. al., "Scaling the Issue Window with Look-Ahead Latency Prediction", in Proc. Int. Conf. on Supercomputing, 2004.

[23] A. Lebeck et. al. A Large, "Fast Instruction Window for Tolerating Cache Misses", in Proc. Intl. Symp. on Computer Architecture, 2002.

[24] P. Michaud, et.al. "Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors", in Proc. of Intl. Conf. on High Perf. Computer Architecture, 2001.

[25] S. Palacharla, N. Jouppi, J. Smith, "Complexity-Effective Superscalar Processors", in Proc. Intl. Symp. on Computer Architecture, 1997.

[26] D.Ponomarev, et.al., "Energy-Efficient Comparators for Superscalar Datapaths", in IEEE Transactions on Computers, July 2004.

[27] D.Ponomarev, et.al., "Energy-Efficient Issue Queue Design", in IEEE Transactions on VLSI Systems, November 2003.

[28] D.Ponomarev, G.Kucuk, K.Ghose, "Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources", in Proc. of Intl. Symposium on Microarchitecute, 2001.

[29] S. Raasch, N.Binkert, S.Reinhardt, "A Scalable Instruction Queue Design Using Dependence Chains", in Proc. Intl. Symposium on Computer Architecture, 2002.

[30] J. Sharkey, D. Ponomarev, "Instruction Recirculation: Eliminating Counting Logic in Wakeup-Free Schedulers", Proc. Euro-Par Conference, 2005.

[31] J. Sharkey, D. Ponomarev, "Non-Uniform Instruction Scheduling", Proc. Euro-Par, 2005.

[32] S. Song, et. al., "The PowerPC 604 Microprocessor", IEEE Micro, 14(5), pp. 8-17, Oct 1004.

[33] S. Srinivasan et. al. "Continual Flow Pipelines", in Proc. Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, 2004.

[34] J. Stark, M Brown, Y Patt, "On Pipelining Dynamic Instruction Scheduling Logic", in Proc. Intl. Symposium on Microarchitecture, 2000.

[35] K. Wilcox et.al.. "Alpha processors: A history of power issues and a look to the future", in Cool-Chips Tutorial, November 1999.

[36] V. Zyuban, "Inhernently Low-Power High-Performance Superscalar Architectures", Ph.D. thesis, University of Notre Dame, 2000.