Energy-Efficient Issue Queue Design

Dmitry V. Ponomarev, *Member, IEEE*, Gurhan Kucuk, *Student Member, IEEE*, Oguz Ergin, *Student Member, IEEE*, Kanad Ghose, *Member, IEEE*, and Peter M. Kogge, *Fellow, IEEE*

***Abstract*—The out-of-order issue queue (IQ), used in modern superscalar processors is a considerable source of energy dissipation. We consider design alternatives that result in significant reductions in the power dissipation of the IQ (by as much as 75%) through the use of comparators that dissipate energy mainly on a tag match, 0-B encoding of operands to imply the presence of bytes with all zeros and, bitline segmentation. Our results are validated by the execution of SPEC 95 benchmarks on a true hardware level, cycle-by-cycle simulator for a superscalar processor and SPICE measurements for actual layouts of the IQ in a 0.18-μm CMOS process.**

***Index Terms*—Bitline segmentation, low-power comparator, low-power instruction scheduling, low-power superscalar datapath.**

I. Introduction

MODERN superscalar datapaths include a number of components for supporting out-of-order execution. In a K-way superscalar processor, instructions are fetched in program order and up to K instructions are dispatched to an issue queue (IQ) also known as the "dispatch buffer," irrespective of the availability of the input operands. As results of prior instructions are computed, they are forwarded to waiting instructions in the IQ. As soon as an instruction waiting in the IQ has all of its input operands available, it becomes ready for execution. As soon as an execution unit ("function unit," FU) is available for a ready instruction, its operands are read out from the IQ into the input latches of the selected FU—a process called instruction issuing—and execution commences.

The IQ is a complex multiported structure that incorporates associative logic in the form of comparators for data forwarding, wakeup logic to identify the instructions ready for execution and additional logic for selecting ready instructions. In aggressive superscalar designs, the size of the IQ ranges from the 20s to more than 100 entries [1]. It is therefore not surprising to see that a significant fraction of the total CPU power dissipation, often as much as 25% [2] is expended within the IQ. Consequently, it is imperative to reduce the power dissipation in the IQ without any significant compromise on the overall performance and the processor's cycle time. The work presented herein does just that.

Manuscript received February 28, 2002; revised October 9, 2002. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under under the PAC-C Program, contract FC 306020020525, in part by the Integrated Electronics Engineering Center, SUNY-Binghamton, and in part by the National Science Foundation under Grant MIP 9504767 and EIA 9911099.

D. V. Ponomarev, G. Kucuk, O. Ergin, and K. Ghose are with the Department of Computer Science, State University of New York at Binghamton, Binghamton, NY 13902-6000 USA (e-mail: dima@cs.binghamton.edu; gurhan@cs.binghamton.edu; oguz@cs.binghamton.edu; ghose@cs.binghamton.edu.

P. M. Kogge is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: kogge@cse.nd.edu).

Digital Object Identifier 10.1109/TVLSI.2003.814321

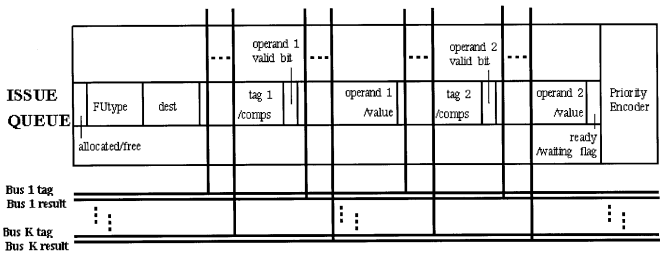

Fig. 1. IQ structure for a datapath with dispatch-bound register reads.

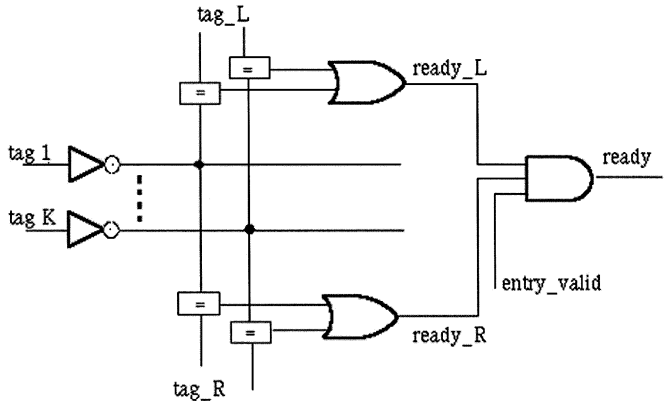

Fig. 2. Wakeup logic of the IQ.

The structure of a typical IQ entry is shown in Fig. 1. As an instruction is dispatched, input registers that contain valid data are read out while the instruction is moved into the allocated IQ entry. As the register values required as an input by instructions waiting in the IQ (and in the dispatch stage) are produced, they are forwarded through forwarding buses that run across the length of the IQ [3]. The IQ entry for an instruction has one data field for each input operand, as well as an associated tag field that holds the address of the register whose value is required to fill the data field. When a function unit completes, it puts out the result produced, along with the address of the destination register for this result on a forwarding bus. Comparators associated with invalid operand slots of valid IQ entries then match the tag values stored in the fields (for waited-on register values) against the destination register address floated on the forwarding bus [3]. On a tag match, the result floated on the bus is latched into the associated input field and the corresponding source operand is marked as valid. Once both operands are marked as valid, the wakeup logic of the IQ (Fig. 2) marks the corresponding instruction as ready for issue. Select logic (not shown here) then

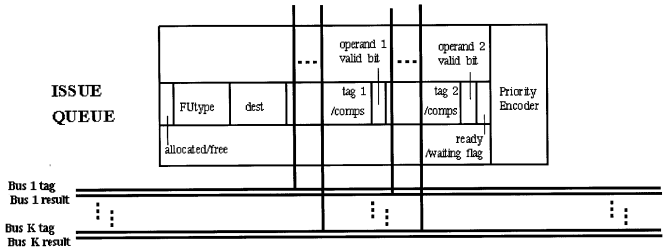

Fig. 3. IQ structure for a datapath with issue-bound register reads.

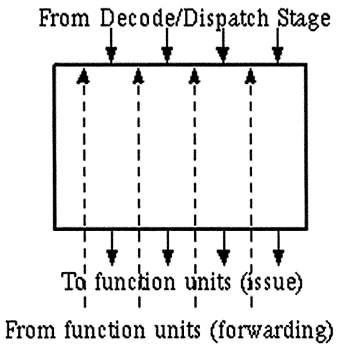

Fig. 4. Blackbox view of the IQ.

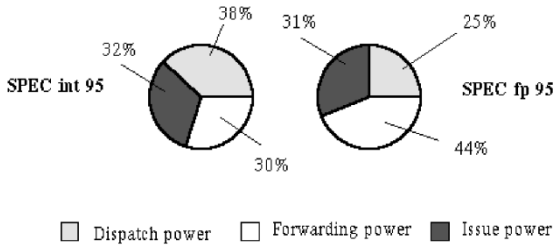

Fig. 5. Energy dissipation components of the traditional IQ for a datapath with dispatch-bound operand reads (% of total).

selects K ready instructions for execution. 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 Pro, IBM Power PC 604, 620, and the HAL SPARC 64.

In an alternative datapath style, register operands are read out at the time of instruction issue. Examples of such datapath are Intel's Pentium 4, Alpha 21 264 and its successors, and MIPS 10 000. In this case, the IQ entry has status bits for each input register in an instruction; register operand values are not part of the IQ entry (Fig. 3). Here, forwarding simply amounts to updating the status of matching input physical registers within established IQ entries. When all input registers of an IQ entry are ready, the corresponding instruction is ready for issue and operands are read out from the physical registers (or bypassed to the FU inputs) as the instruction issues.

Fig. 4 depicts the blackbox view of an IQ as described above. This is essentially a multiported register file with additional logic for associative data forwarding from the forwarding buses and associative addressing logic that locates free entries and entries ready for issue. We assume a four-way superscalar processor for our studies. The IQ is assumed to have four read ports, four write ports and four forwarding buses. The four write ports are used to establish the entry for up to four instructions simultaneously in the IQ at the time of dispatching. The four read ports are used to select up to four ready instructions for issue and move them out of the IQ to the FUs. If the IQ, shown in Fig. 4, has N entries and each instruction can have up to two source operands, then eight comparators are needed for each IQ entry (for a total of $N * 8$ comparators for the entire IQ). This is because each source operand can be delivered through any of the four result/tag buses. The main sources of energy dissipation in the IQ are as follows:

1) Energy dissipated in the process of establishing IQ entries for dispatched instructions: in locating a free entry associatively and in writing into the selected entry.
2) Energy dissipated when FUs complete and forward the results and/or status information to the IQ entries. A significant fraction of this energy dissipation is due to the tag comparators used for associative matching to pick up forwarded data.
3) Dissipations at the time of issuing instructions to the FUs: in arbitrating for the FUs, enabling winning instructions for issue and in reading the selected instructions and their operands from the IQ.

In this paper we examine the use of several techniques for reducing these dissipations. For this purpose, power measures are obtained using detailed cycle-by-cycle, true hardware level simulations of the SPEC 95 benchmarks, and the use of SPICE measurements of actual 0.18-μm layouts of the IQ.

In an earlier paper [4], we examined a technique for splitting up an IQ into distributed IQs tailored for specific instruction types and the suppression of leading zeros in operands to save power. As wire delays become significant, such a distribution may impact the cycle time adversely. We focus this paper on techniques that avoid such a distribution and maintain the centralized structure of the IQ, with minimal impact on the circuit design and layouts. This work is an extended version of our earlier paper [5].

The rest of the paper is organized as follows. Section II describes the proposed mechanisms for power reduction in the IQ followed by our simulation methodology in Section III. Section IV presents our simulation results. Section V reviews the related work and we conclude in Section VI.

II. Reducing Issue Queue Power: Three Approaches

We examine the use of three techniques for reducing the IQ power dissipations. To set the right context, it is useful to examine the major power dissipation components within the IQ. Fig. 5 shows the total IQ energy breakdown across the three components. Results were obtained using our simulation framework, which is described in Section III. The distribution is shown for a datapath with dispatch-bound operand reads, that is, wide IQ is in use. The numbers represent the averages over the SPEC 95 integer and floating-point benchmarks. Forwarding power represents a relatively higher percentage of the

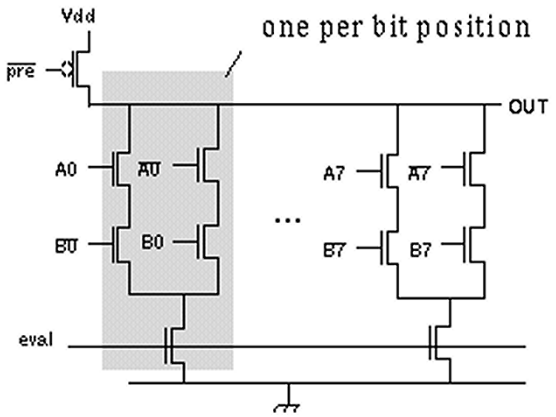

Fig. 6. Traditional pull-down comparator.

total-power dissipation for floating point benchmarks, because more tag comparisons are performed in an average cycle for two reasons: first, more tags are broadcast because floating point benchmarks exhibit higher instruction-level parallelism (ILP) than integer benchmarks and second, more slots in the IQ await for results in the course of execution of floating point benchmarks. Simulation of SPEC 95 benchmarks within our experimental framework shows that on the average about 21 slots are waiting for the results during the execution of integer benchmarks and 25 slots are waiting during the execution of floating point benchmarks (for this graph, we assumed that only the comparators associated with the invalid sources of valid entries are activated; in the result section we also consider the case when all comparators are active). All such slots employ the traditional pulldown comparators (that dissipate energy on tag mismatches) in current implementations of IQs [3].

To reduce the energy spent in data forwarding directly, we explore the use of comparators that dissipate energy predominantly on a tag match. We then explore the use of 0-B encoding [4, 6, 7, 8] to reduce the number of bitlines activated during dispatching, forwarding, and issuing. Finally, we add bit-line segmentation to reduce energy dissipations in the bitlines that are driven. The overall energy saving realized for the IQ by using all of these techniques in combination is between 46% and 75%, and this is realized with an acceptable increase in the cycle time of the processor and silicon real estate for the IQ.

A. Using Energy-Efficient Comparators in the IQ

The typical comparator circuitry used for associative matching in an IQ is a dynamic pulldown comparator or an 8-transistor associative bitcell. This is depicted in Fig. 6. Such comparators have a fast response time to allow matching and the resulting updates to be completed within the cycle time of the processor. All of these comparators dissipate energy on a mismatch in any bit position. A significant amount of energy is thus wasted in comparisons that do not locate matching entries, while little energy (in the form of precharging) is spent in locating matching entries. In a typical cycle, only a small percentage of the slots waiting for the result actually match the destination address of the forwarded data. As an example, the data collected for the simulated execution of SPEC 95 benchmarks on our system indicates that about 23-operand slots out

TABLE I
ISSUE QUEUE COMPARATOR STATISTICS

Number of bits matching ⟹	% of total cases			
	2 LSBs	4 LSBs	6 LSBs	All 8 bits
Avg. SPECint 95	27.2	9.8	5.7	5.6
Avg. SPECfp 95	33.1	10.7	5.4	4.4
Avg. all SPEC 95	30.5	10.3	5.6	4.9

of the 128 that we have in our 64 entry IQ are actually waiting for results (i.e., the comparators for these slot are enabled for comparison). Out of these, only one to four comparators produce a match per cycle on the average. This is clearly an energy-inefficient situation, as more energy is dissipated due to mismatches (compared to the number of matches) with the use of traditional comparators that dissipate energy on a mismatch. We propose to remedy this by designing and using comparators that (predominantly) dissipate energy on a match.

In theory, one can design CMOS comparators to dissipate energy dynamically only on a full match but these designs require a large number of transistors and/or switch slowly. Instead of choosing such a design, we opted for a comparator that dissipates minimum energy on (infrequent) partial mismatches in the comparand values and dissipates an acceptable amount of energy on a full match. This comparator consciously takes into account the characteristics of the distribution of the physical register addresses that are compared in a typical IQ to minimize energy dissipations for partial matches.

Table I shows how comparand values are distributed and the extent of partial matches in the values of two adjacent bits in the comparands, averaged over the simulated execution of SPEC 95 integer and floating point benchmarks, as well as the average over both the integer and floating point benchmarks. The lower-order 4 bits of both comparands are equal in roughly 10.3% of the cases, while the lower-order 6 bits match 5.54% of the time. A full match occurs 4.93% of the time on the average. Equivalently, a mismatch occurs in the lower-order 4-bits of the comparator 89.7% of the time. The reason for the behavior depicted in Table I is that the destination physical registers are typically assigned to neighboring instructions from the localized regions. If, for example, the reorder buffer (ROB) slots are used to implement physical registers, the slots are allocated to instructions consecutively in program order. If a separate physical register file is used instead, the search for a free physical register is performed starting from the lower addresses, thus the localized allocations result in this case as well. This implies that the majority of comparisons within the IQ are between the neighboring register addresses and, therefore, the bulk of partial matches occur in the most significant bit positions. Our comparator design directly exploits this fact by limiting dynamic energy dissipation due to partial matches to less than 10.3% of cases when the lower-order 4- and 6-bits match; no energy dissipation occurs in the more frequent cases of the higher-order bits matching. The overall energy dissipation due to partial mismatches is thus greatly reduced. Energy dissipation occurs, of course, on a full match but this dissipation is smaller than that of comparable traditional comparators (that pulls down a

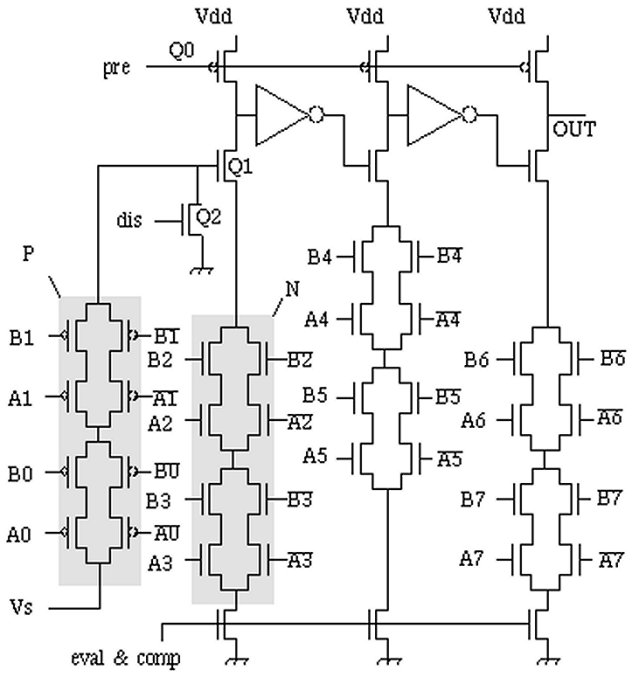

Fig. 7. The proposed comparator.

precharged line only on a mismatch in one or more bit positions).

Fig. 7 depicts our proposed comparator for comparing two 8-bit comparands, A7A6..A0 and B7B6..B0, where 0 is the least significant bit. The basic circuit is a three-stage domino logic, where the first stage detects a match of the lower 4 bits of the comparands. The following two stages do not dissipate any energy when the lower-order 4-bits do not match. The precharging signal is cutoff during the evaluation phase (when evaluation is active) and an evaluation signal is applied to each stage of the domino logic only when the comparison is enabled (comparison is high). The series structure P composed of eight p-transistors passes on a high voltage level (Vs, where Vs is lower than Vdd but higher than the lower threshold for a logic 1 level) to the gate of the n-transistor Q1 only when the two lower-order bits of the comparands match. The series structure N turns on when the next pair of lower-order bits of the comparands (A3A2 and B3B2) match. When comparison is enabled, the output of the first stage (driving an inverter) goes low during the evaluation phase only when all lower-order 4-bits of the comparands match. Until such a match occurs, no dynamic energy is dissipated in the other stages of the comparator. Transistor Q2 is needed to prevent Q1 from turning on due to the presence of charge left over on its gate from a prior partial match of the two lower-order bits. The charge moved from the gate of Q1 by Q2 is dissipated to ground only when there is a subsequent match in bits 3 and 2 (which turns the structure N on). This effectively reduces dissipations in the case when only the two lower-order bits match; this dissipation could be further minimized by keeping Vs slightly lower than Vdd, but we did not explore this option because of the implementation difficulties. As in any series structure of n-transistors that pull down a precharged line, the W/L ratios of the n-devices go up progressively from the top to the bottom (ground). The p-transistors in the structure P, the precharging transistors and the inverters are sized to get an acceptable response time on a match.

The comparator delay component that lies on the critical path in a typical cycle is the evaluation delay. The precharge time can be used to provide and stabilize the new input data, such that the inputs are ready when the evaluation signal rises. For example, if the comparators are used within the IQ, then the source operand tags (which are the data inputs to the comparators) can be driven across the IQ while the comparator is precharged. Therefore, the evaluation time defines the delay of both comparators for all practical purposes.

For the proposed comparator, a separate discharge signal (discharge in Fig. 7) is used to discharge the gate of the n-transistor Q1 to prevent false matches. Notice, that the discharging action is not on the critical path, since the inputs can still be changing when discharge signal is high. This may lead to a momentary short-circuit energy dissipation, but this energy is small and was fully accounted for in our SPICE measurements. The evaluation delay of the comparator of Fig. 7 in a full-match condition is comprised of five nonoverlapping components, which are listed as follows:

1) T1—time needed to discharge the input of the first inverter;
2) T2—delay of the first inverter;
3) T3—time needed to discharge the input of the second inverter;
4) T4—delay of the second inverter;
5) T5—time needed to discharge the output.

To compare the evaluation delay of the proposed comparator with that of the traditional design, we first optimized the traditional comparator circuitry by minimizing its delay through careful resizing of the devices. The evaluation delay is highest when the comparands mismatch in only one bit position, since the matchline is discharged through a single stack of n-devices. This is the situation that we considered in our experiments. There is an optimal width of n-devices that results in the minimum evaluation delay. As the width of n-devices increases, the output node discharges faster, but the output capacitance also increases, resulting in slower discharging if the width is increased beyond a certain point. The minimum evaluation delay of the traditional comparator determined by the SPICE simulations was 121 ps.

The evaluation delay of the proposed comparator increased by 150 ps compared to the traditional design. The breakdown across the five delay components is as follows: T1 = 126 ps, T2 = 37 ps, T3 = 41 ps, T4 = 39 ps, and T5 = 28 ps. Notice that T3 and T5 are significantly lower than T1 because the N-blocks discharge simultaneously during the first 126 ps. The time T1 also includes the propagation delay of the high voltage level Vs through the P-block to the gate of n-transistor Q1 (when bits 0 and 1 match). Most of this delay is overlapped with the discharging of the source of Q1 through the first N-block (when bits 2 and 3 match). Fig. 8 shows the waveforms obtained from the SPICE simulations (using the Cadence toolkit) of the proposed comparator. The waveforms are shown for the situation of a full match in the comparands. Fig. 8(a) depicts the voltage at the output node of the comparator as a function of time. As shown, the output discharges after 271 ps, which it should in a full match situation. Fig. 8(b) shows the voltage at the output of

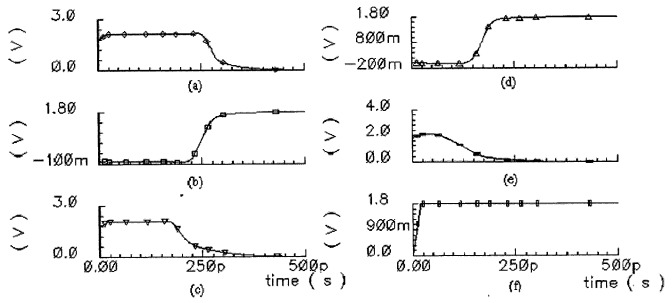

Fig. 8. SPICE waveforms of the proposed comparator.

the second inverter, Fig. 8(c) depicts the voltage at the input of the second inverter, Fig. 8(d) shows the voltage at the output of the first inverter, Fig. 8(e) shows the voltage at the input of the first inverter and, finally, Fig. 8(f) depicts the evaluation signal.

For the cycle time of 566 ps (which was the read access time of the multiported IQ used for the datapath with issue-bound operand reads in our simulations), the increase in comparator evaluation delay represents an overhead of about 26%. The read cycle of the IQ used for the datapath with dispatch-bound operand reads is 658 ps, and the increase in cycle time due to the use of the new comparator is about 23%. We discuss the timing considerations in more detail in Section II-C. An alternative dissipate-on-match comparator circuitry which is actually slightly faster than the traditional comparator was introduced in our recent work in [20].

The data inputs to the comparator of Fig. 7 (bits A0–A7, B0–B7, and their complements) are driven after buffering at the beginning of a clock cycle). Because of this, the sizing of the transistors within the P-block and the use of a Vs value close or equal to Vdd, circuit malfunction due to charge sharing among the P-chain, the gate of Q1 and Q2 is avoided. Similar malfunctions due to charge sharing among the n-devices in the pulldown stack are also avoided by progressively increasing the sizes of the n-transistors in each stack as mentioned earlier. Since the outputs of all enabled comparators are latched at the end of every cycle, keeper devices are not needed within the comparator. A pleasant side effect of the use of the comparator of Fig. 7 within an IQ has to do with current surges (di/dt). With the use of traditional comparators, most of the comparators draw a significant amount of energy from the power supply in every cycle as very few comparators match. With the use of the comparators of Fig. 7, where only the few matching comparators draw energy from the power supply in each cycle, the di/dt spike is drastically reduced.

The area overhead of the proposed comparator is 28% (642 μm^2 versus 500 μm^2 for the traditional design). To conclude the discussion of the proposed comparator, Tables II and III shows the energy dissipations of the traditional and the proposed comparators for various matching patterns in the comparands. Combined with the performance simulation results summarized in Table I and the relevant percentages for the traditional comparator (not shown here), these numbers were used to estimate the energy savings.

B. Using 0-B Encoding

A study of data streams within superscalar datapaths as reported in [8] shows that significant number of bytes within

TABLE II
ENERGY DISSIPATIONS OF THE TRADITIONAL COMPARATOR FOR VARIOUS MATCHING PATTERNS

Number of matching bits in the comparands	Energy of the traditional comparator (fJ)
0	709.2
1	671.0
2	631.7
3	593.1
4	551.3
5	513.2
6	470.8
7	432.7
8	7.7

TABLE III
ENERGY DISSIPATIONS OF THE NEW COMPARATOR FOR VARIOUS MATCHING PATTERNS

Number of matching leading bits in the comparands	Energy of the new comparator (fJ)
None of the bits	8.2
2 least significant bits	142.5
4 least significant bits	473.8
6 least significant bits	792.0
Full match	889.1

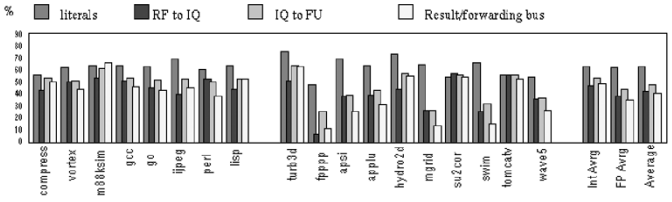

Fig. 9. Percentage of zero bytes on various paths within the superscalar processor.

operands on most of the flow paths (dispatch stage to IQ, IQ to function units, function units to destinations and forwarding buses, etc.) contains all zeros. On the average, in the course of simulated execution of SPEC 95 benchmarks on cycle accurate and true register level simulator, about half of the byte fields within operands are all zeros. This is really a consequence of using small literal values, either as operands or as address offsets, byte-level operations, operations that use masks to isolate bits etc. Considerable energy savings are possible when bytes containing zeros are not transferred, stored, or operated on explicitly. The same observation was made for caches [9], function units [6], and scalar pipelines [7].

Fig. 9 shows the percentage of bytes with all zeros on various paths within the superscalar processor simulated as described in Section III. On the average across all benchmarks, about 63% of bytes in the literals that are written to the IQ directly from the instruction register are the bytes containing all zeros. Within the data items being read from the register file to the IQ, 42% of all

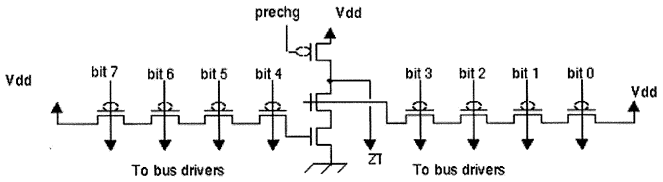

Fig. 10. Encoding logic for all zero bytes.

bytes are the bytes that contain all zeros. On the issue path, 48% of all bytes are zero bytes. On this path, the data stream represents a mixture of literals (with high percentage of zero bytes) and the regular data items (with lower percentage of zero bytes), and therefore the resulting percentage of zero bytes on this path is higher than on the path from the register file but lower than the percentage of zero bytes in literals. Finally, about 41% of all bytes traveling on the result/forwarding busses are zero bytes. Notice that across the individual benchmarks the lowest percentage of zero bytes was observed for $fpppp - 6\%$ on the path from the register file to the IQ and 11% on the forwarding/result path. In fact, $fpppp$ is the only SPEC95 benchmark, for which the 0-B encoding logic described in this section results in higher energy dissipation in certain cases. The statistics of Fig. 9 was collected only for the bytes with significant data. For example, if a 32-bit data piece is transferred, we only consider the 4 bytes that constitute these 32 bits. For a 64-bit piece of data we analyzed all 8 bytes.

By not writing zero bytes into the IQ at the time of dispatch, energy savings result as fewer bitlines need to be driven. Similarly, further savings are achieved during issue by not reading out implied zero valued bytes. This can be done by storing an additional bit with each byte. This bit indicates if the associated byte contains all zeros or not. The contents of this zero indicator bit can be used to disable the word select strobe from going to the gates of the pass transistors. By controlling the sensitivity of the sense amps, we can also ensure that sense amp transitions are not made when the voltage difference on differential bitlines is below a threshold (as would be the case for the precharged bitline pairs associated with the bitcells whose readouts are disabled as described above). Zero-valued bytes do not have to be driven on the forwarding buses that run through the IQ—this is another source of energy savings that result from 0-B encoding.

Fig. 10 depicts the circuitry needed to derive the zero indicator bit (ZIB) for a byte; such an encoder can be implemented within function units, decode/dispatch stage of the pipeline or within output latches/drivers. Two n-devices (not shown) are also used to discharge any charge accumulated on the gates on the two pulldown n-transistors to prevent false discharges of the output due to prior zero indications. The logic used to disable the readout of bytes with all zeros is shown in Fig. 11. P-transistor Q4 and n-transistor Q5 are used to connect the read line of the corresponding byte to the word select line if the value stored in the ZIB is one. Otherwise, the local read line is grounded and the byte is not read out. The n-transistor Q5 in parallel with the p-transistor Q4 is needed to ensure that the local word select line for a byte is properly and timely discharged at the end of the read cycle. Effectively, the p-transistor Q4 and the n-transistor Q5 formed a complementary switch to guarantee fast assertion

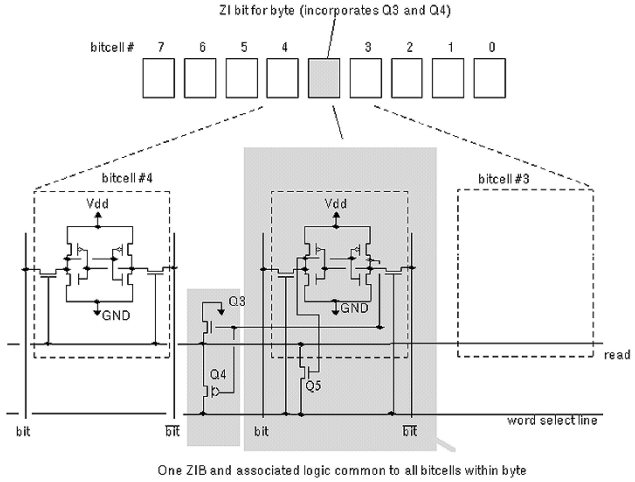

Fig. 11. Bit-storage enhancements for avoiding the reading of all-zero bytes (for a single port).

and deassertion of the local word select line of a no n-zero byte. To further limit the cycle time increase due to the use of ZIB logic, we used such pair of p- and n-transistors for every 4 bits, therefore requiring the use of $2n$ and $2p$ transistors for each byte. This increased the area of the IQ and the device count, but resulted in significantly faster operation of the circuit. In our layouts, we also matched the height of a bitcell incorporating 0-B encoding logic to that of the traditional bitcell by using the additional metal layer (metal 4 in this case). Thus, only the width of the IQ increased as a result of using 0-B encoding logic.

C. Area and Timing Considerations

The price paid for energy savings within the IQ through the use of 0-B encoding is in the form of an increase in the area of the IQ by about 21% for the datapath with dispatch-bound operand reads and 17% for the datapath with issue-bound operand reads. The former number is higher, because more bytes are being zero-encoded if the operand results are stored in the IQ. The increase in the number of RAM cells used to implement the IQ results in slightly higher static energy dissipations—this is in the order of 10%. There is also some increase in the IQ access time, which we will now quantify. In the baseline IQ, the read access can be performed in 566 ps (from the appearance of the address at the input to the driver of the decoder to the appearance of the data at the output of the sense amp) for the issue-bound datapath. The use of the ZI bit to enable the corresponding byte for readout increases this delay to 616 ps, which represents just less than 9% increase in cycle time. For the dispatch-bound datapath, the data can be read from the baseline IQ in 658 ps. In the design with zero encoding logic, the delay increases to 740 ps, thus stretching the cycle by about 12%.

Notice that the additional delays introduced by the comparators and the 0-B encoding logic are not cumulative. While comparators are used in the wakeup stage, the 0-B encoding logic is used during the select cycle in the course of reading the selected instructions out of the IQ. Since wakeup and select operations are spread out over two pipeline stages in high-frequency de-

signs, the overall increase in the pipeline cycle time is the maximum of the two overheads. If the wakeup cycle determines the critical path, then the comparator design proposed in this paper is not an attractive choice and other solutions, such as the ones proposed in [20] should be used instead. However, if a small slack exists in a stage where comparators are used (the wakeup stage), then the comparator of Fig. 7 can be used without any effect on the cycle time. For example, consider the use of the proposed comparator in conjunction with 0-B encoding logic for a dispatch-bound datapath. The zero-encoding logic increases the duration of the select cycle by about 80 ps while the new comparator increases the wakeup cycle by about 150 ps. So, if a slack of as little as 70 ps exists in the wakeup stage, the increase in the processor's cycle time will be bounded by the delays of zero-encoding logic and not by the delays of the comparator.

D. Using Bitline Segmentation in the IQ

Bitline segmentation is just one of the many techniques used to reduce power dissipation in RAMs [10]–[12]. In this section, we show how to incorporate bitline segmentation into our low power IQ design.

As mentioned earlier, the IQ is essentially a register file with additional associative logic for data forwarding. Writes to the IQ, which occur at the time of dispatch, use the normal logic for a register file to set up the IQ entry for dispatched instructions. For each instruction dispatched in a cycle, a write port is needed. The only difference from a normal register file is that the word being written to from each write port is selected associatively (an associative search is needed to locate free entries, i.e., words within the IQ), instead of being selected through an explicit address. At the time of instruction issue, instructions ready for execution are read out from the IQ through independent read ports. Other than the use of the wakeup and arbitration logic to select ready entries, this readout is identical to what happens when a normal register file is read out. Sense amps, similar to those used in RAMs are needed to sense the data being read out as the bitlines are fairly long. As in a multiported RAM or register file, the bitlines in the IQ are a significant source of energy dissipation in the course of instruction dispatching (writes) and instruction issuing (reads). The bitlines associated with each read and write port present a high capacitive load, which consists of a component that varies linearly with the number of rows in the IQ. This component is due to the wire capacitance of the bitlines and the diffusion capacitance of the pass transistors that connect bitcells to the bitlines.

The capacitive loading presented by the bitlines in the IQ can be reduced through the use of bitline segmentation. The entire IQ is viewed as a linear array of segments for this purpose, with consecutive bitcell rows making up a segment. Fig. 12(a) is useful in understanding how bitline segmentation reduces the capacitive loading encountered during reads and writes from the bitline. As an example, consider an IQ with 64 rows, which has been restructured into four segments. Each segment will then consist of 16 consecutive rows. The original bitline, shown in the left of Fig. 12(a) is loaded by the diffusion capacitance of 64 pass devices, the diffusion capacitances of the precharging and equlibrator devices, sense amp input gate capacitances and the diffusion capacitances of tri-stated devices used to drive the

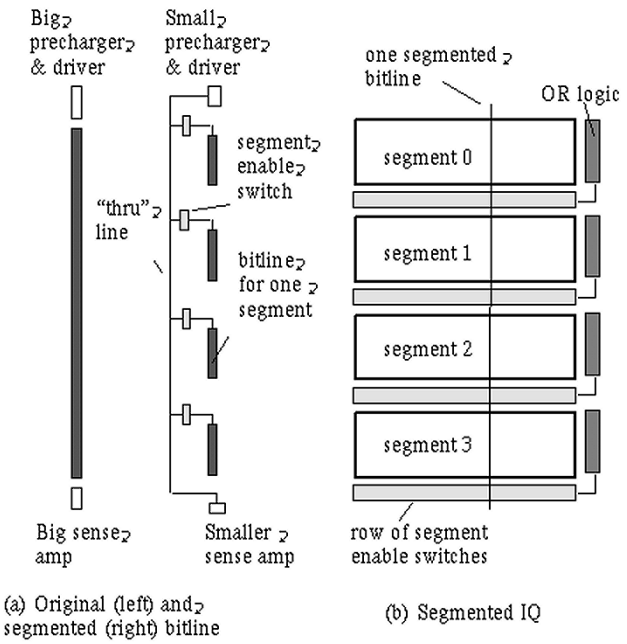

Fig. 12. Bitline segmented IQ.

bitlines (during a write). In addition, there is the wire capacitance of the bitline itself. In the segmented version, the bitline is split into four segments; each segment of the bitline covers a column of the bitcells of the rows within a segment. As a result, the capacitive loading on each segment is lowered: each segment is connected to only 16 pass devices and the wire length of the bitline segment is one fourth of the original bitline.

To read a bitcell within a segment, the bitline for that segment has to be connected to the precharger and sense amp; for a write, the bitline segment has to be connected to the tri-state enable devices for the bitline driver. This is accomplished by running a "through" wire across all of the segments (typically in a different metal layer, right over the segmented bitlines), which is connected to the prechargers, sense amp and tri-state drivers as in the nonsegmented design. A switch is then used to connect the segment in question to this "through" line. For the IQ, the segment switch is turned on by OR-ing the associative read or write enable flags for the port (associated with the bitline) for all the rows in that segment. As the effective capacitive loading on the through line and the connected bitline segment is smaller than the capacitive loading of the unsegmented bitline, lower energy dissipation occurs during reads or writes. The extent of energy savings depends on the relative contribution of the energy dissipated in the bitlines to the total energy expended within the IQ. On the down side, additional energy dissipation occurs in the control logic for the segment enable switch, the energy needed to drive the switches for all of the columns and the loading imposed on the through line by the diffusion capacitances of the complementary segment enabling switches. By carefully choosing the size of a segment, the overall energy dissipations can be minimized—making the size of a segment too small can actually increase the overall energy consumption, while making the size too large defeats the purpose of segmentation. An optimal segment size of eight rows was discovered for

TABLE IV
ARCHITECTURAL CONFIGURATION OF A SIMULATED FOUR-WAY SUPERSCALAR PROCESSOR

Parameter	Configuration
Machine Width	4-wide fetch, 4-wide issue, 4-wide commit
Window Size	64 entry issue queue, 100 entry reorder buffer, 32 entry load/store queue, 128 Int phys. reg, 128 FP phys.reg.
L1 I-Cache	32 KB, direct-mapped, 32 byte line, 2 cycles hit time
L1 D-Cache	32 KB, 4-way set-associative, 32 byte line, 2 cycles hit time
L2 Cache combined	512 KB, 4-way set-associative, 64 byte line, 4 cycles hit time
BTB	1024 entry, 2-way set-associative
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
Function Units 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)

the 64-entry IQ described here. Fig. 12(b) depicts a segmented IQ and shows that there is some increase in the overall area of the IQ due to the use of a row of segment enabling switches with each segment.

III. EVALUATION METHODOLOGY

To evaluate the power savings achieved in the IQ by the use of the proposed mechanisms, we used the AccuPower toolset [13]. The widely-used Simplescalar simulator [14] was significantly modified (the code for dispatch, issue, writeback and commit steps was written from scratch) to implement true hardware level, cycle-by-cycle simulation models for such datapath components as the IQ, the reorder buffer, physical register files, architectural register files and the rename table. Configuration of the simulated system is shown in Table IV. The execution of the SPEC 95 benchmarks was simulated (each benchmark was run for 200 million instructions after a 200 million instructions startup phase). Benchmarks were compiled using the Simplescalar GCC compiler that generates code in the portable ISA (PISA) format. Reference inputs were used for all the simulated benchmarks.

Detailed accounting for detecting zero bytes in operands and lower-level transition counting was implemented by a separate thread. Transition counts for reads, writes, associative addressing, FU arbitration, tag matching, data latching, and other notable events were recorded separately. Transition counts and other data gleaned from the simulator were then fed into a power estimation program that used dissipation energies measured using SPICE for actual 0.18-μm layouts of key datapath components. (The process used was a 0.18-μm six metal-layer CMOS process, TSMC). Our power estimation program generated power measures in picojoules for major energy dissipating events within the IQ for each benchmark in the SPEC 95 suite individually as well as for the averages of the integer and floating point benchmarks and total averages.

IV. RESULTS

In this Section, we estimate the power savings that can be achieved within the IQ if the three techniques described in this paper are used. The extent of achievable power savings, as well as the relative contribution of each technique to the total power reduction depends on the IQ design style and whether all comparators are activated during the tag matching operation or only those comparators that are associated with the slots awaiting a result are activated. We now analyze the four possible combinations in detail. We begin with the IQ of a datapath with dispatch-bound operand reads.

Fig. 13 shows the total energy reduction within the IQ if all three techniques discussed in this paper are used. Here, we assumed that forwarding comparisons are enabled only for the IQ entries that are awaiting a result; comparisons are not done for unallocated entries or allocated entries that have already been forwarded the result. Bitline segmentation results in 24% of energy reduction on the average, with the bulk of the savings coming from reduced bitline dissipations during writes to the IQ at the time of instruction dispatching. Specifically, the dispatch power is reduced by more than 66%. Effects of the bitline segmentation are much less noticeable during reads from the IQ, because this component of energy dissipation is dominated by the energy of sense amps. As the dispatch energy is a sizable portion of the total energy in this design, the overall energy savings due to the use of bitline segmentation are quite significant. Zero-byte encoding by itself reduced the IQ energy by about 24%—this is not shown in the graph. The energy expended during instruction dispatching is reduced by 24%, during instruction issuing by 32% and the fact that fewer forwarding data lines need to be driven also reduces the forwarding energy by about 16%. To summarize, bitline segmentation and 0-B encoding are equally effective at reducing the IQ energy. The combination of 0-B encoding and bitline segmentation results in 42% energy reduction in the IQ. Notice that the combined energy reduction is a little smaller than the sum of the individual reductions, because 0-B encoding is applied to the bitlines with the reduced capacitive load (due to the use of segmentation). Finally, the last bar of Fig. 13 shows the energy reductions is the IQ possible when all three techniques are used in concert. The new comparator is 75% more energy-efficient on the average than the traditional comparator. However, in this configuration the forwarding energy is dominated by the energy of driving the tag and the data lines across the IQ and not by the energy dissipated within the comparators. The energy savings in the forwarding component amount to 12% is the proposed comparators are used. In terms of total IQ energy, the savings due to the use of the new comparators amount to only 4%. The combined effect of all three techniques is 46% energy reduction in the IQ—this is shown in the last bar of Fig. 13.

If spurious comparisons within the IQ are performed, then the energy dissipated in the course of tag comparisons becomes a dominant source of dissipations. Fig. 14 shows the energy savings achievable by the use of our three techniques in this case. For this analysis, we assumed that for each result tag that is driven across the IQ, all comparators associated with that particular tag bus are activated. Thus, 128 comparisons are performed

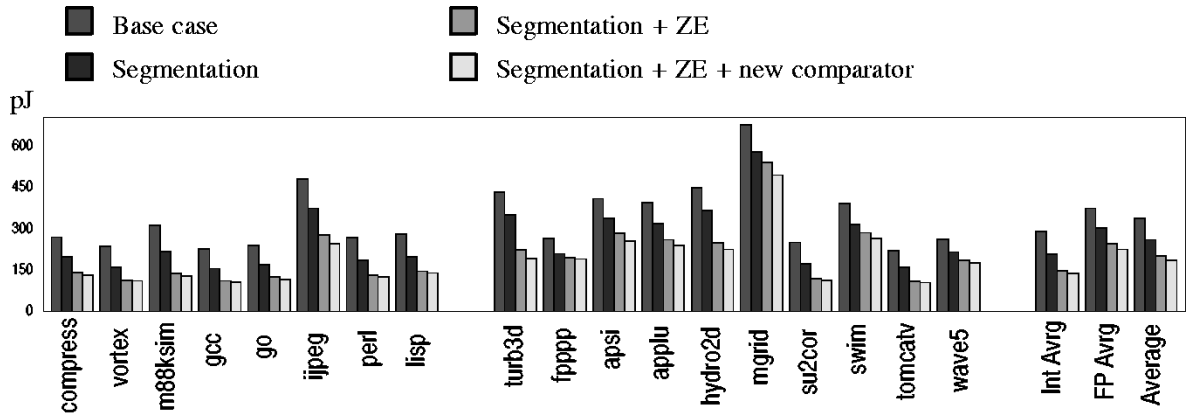

Fig. 13. Total power dissipation within IQ using bitline segmentation, power efficient comparators and 0-B encoding for the datapath with dispatch-bound register reads. Comparisons are only performed for the IQ slots that wait for a result.

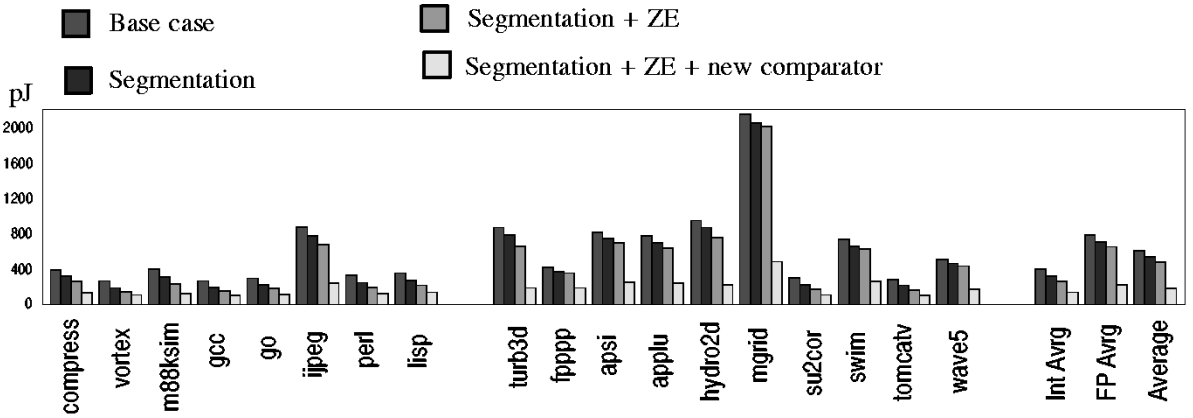

Fig. 14. Total power dissipation within IQ using bitline segmentation, power-efficient comparators and 0-B encoding for the datapath with dispatch-bound register reads. Comparisons are performed for all IQ slots irrespective of whether they wait for a result.

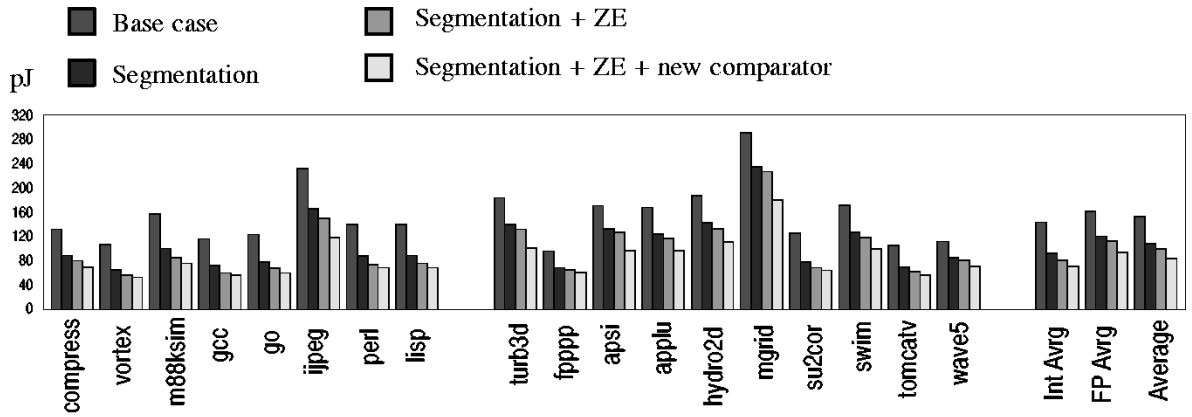

Fig. 15. Total power dissipation within IQ using bitline segmentation, power efficient comparators and 0-B encoding for the datapath with issue-bound register reads. Comparisons are only performed for the IQ slots that wait for a result.

for each broadcasted tag irrespective of how many IQ slots are actually waiting for a result. In such a system, the forwarding energy represent about 66% of total energy dissipated in the IQ, and the large fraction of this energy is attributed to the comparators. Therefore, the use of dissipate-on-match comparators results in the highest energy reduction compared to the use of other techniques—38% on the average across all benchmarks. The benefits of bitline segmentation are limited to 16.5%, as it has no impact on the forwarding energy. The energy reduction due to the use of 0-B encoding is 17%—again, this is significantly less than in the case shown in Fig. 13. The combined effect of bitline segmentation and 0-B encoding is 28.5% energy reduction in the IQ. The total energy savings in the IQ from the use of the three techniques in combination is in excess of 66% on the average across all SPEC 95 benchmarks.

We now present similar results for the IQ used in datapaths with issue-bound operand reads. Fig. 15 depicts the power savings achievable within the IQ, assuming that only useful tag

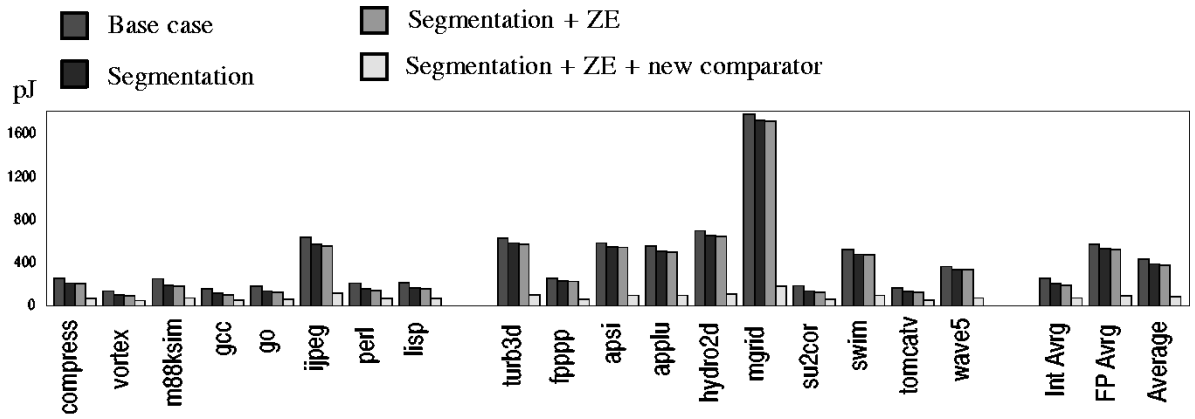

Fig. 16. Total power dissipation within IQ using bitline segmentation, power efficient comparators and 0-B encoding for the datapath with issue-bound register reads. Comparisons are performed for all IQ slots irrespective of whether they wait for a result.

comparisons are performed. Here, the use of bitline segmentation results in energy reduction of 30%, savings achieved by the use of 0-B encoding are in the order of 15%, and the combined energy reduction of the two techniques is 37%. The proposed comparator reduces the forwarding energy by 25% and the overall energy of the IQ by 9%. The combined effect of the three techniques is 46% reduction in the IQ energy dissipations. If we compare the results of Fig. 13 and Fig. 15, the relative contribution of each technique varies, but the overall achievable energy savings are in the order of 46% for both designs.

Finally, in Fig. 16 we show the energy savings realizable within the IQ of a datapath with issue-bound operand reads if spurious tag comparisons are allowed. The resulting energy savings are as follows: bitline segmentation—16%, 0-B encoding—8%, the combination of bitline segmentation and 0-B encoding—20%. The use of the proposed comparators results in 55% of the total IQ energy reduction—a very significant savings attributed to the fact that tag comparisons represent a major percentage of the total IQ power budget in this design—in excess of 60%.

Combined, the three techniques result in 75% of energy reduction in the IQ, if spurious comparisons are allowed.

To summarize, the importance of each considered power reduction technique depends on the datapath design style (dispatch-bound register reads versus issue-bound register reads) and on whether spurious comparisons are performed or not. The combined power savings in the IQ range between 46% and 75% for various design alternatives.

V. Related Work

The problem of power reduction in dynamically scheduled superscalar processors in general, and in the IQs in particular, has received a considerable attention in recent literature. One approach to power minimization in the IQs has to do with dynamic resizing of the queue based on the requirements of the application. Several resizing techniques have been proposed along these lines. Specifically, in [15], Buyuktosunoglu *et al.* explored the design of an adaptive IQ, 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 [2], Folegnani and Gonzalez introduced a FIFO IQ 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 IQ 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 IQ. 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. In [16], we introduced a resizing mechanism, where the IQ occupancy (measured as the number of valid entries within the IQ) was used as an indication of resource needs to drive the IQ resizing, in conjunction with similar resizing of the reorder buffer and the load/store queue.

In [17], Bahar and Manne studied the multiclustered processor with replicated register files. The dispatch rate was varied between 4, 6, and 8 to allow an unused cluster of function units (including percluster IQs) 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. Significant power savings within the dynamic scheduling components were achieved with a minimum reduction of the IPC.

In [18], Brooks *et al.* suggested that the use of comparators that dissipate energy on a tag match could significantly reduce the power dissipated in the IQ. The actual designs of the comparators and the extent of achievable power savings were, however, not presented. In this paper, we proposed a design of a dissipate-on-match comparator and quantified the power savings realized by using such comparators within the IQ. Alternative comparator designs were also proposed in our recent work in [20].

Two recent efforts [3], [19] have attempted to reduce the complexity of the IQ. The implications of additional peripheral logic on the overall power dissipation was not addressed. In [3], Palacharla *et al.* proposed an implementation of the IQ through several FIFOs, so that only the heads of each FIFO need to be

examined for issue. This significantly reduced the number of instructions that need to be considered for issue every cycle. The scheme of [19] reduces the amount of associative lookup in the issue process by limiting it only to the instructions within the small out-of-order queue. This results in significant reduction of the complexity of the control logic, but the effect on power dissipation is not clear, as the additional structures used in [19] certainly have some power overhead.

A related effort is the work of [22], where Huang *et al.* proposed to save energy in the IQ by using indexing to only enable the comparator at the single instruction to wake up. In rare cases, the mechanism reverted back to the usual tag broadcasts, when more than one instruction needed to be awakened.

In [21], Ernst and Austin explored the design of the IQ with reduced number of tag comparators. They capitalized on the observation that most instructions enter the instruction window with at least one of the source operands ready. These instructions were put into specialized entries within the IQ with only one (or even zero) comparators. The authors of [21] also introduced the last tag speculation mechanism to handle the instructions with multiple unavailable operands. The combination of the two techniques reduced the number of tag comparators used by 75% compared to the traditional IQ designs.

VI. Concluding Remarks

We studied three techniques to reduce the energy dissipation in the instruction IQs of modern superscalar processors. First, we proposed the use of comparators in forwarding/tag matching logic that dissipate energy mainly on the tag matches. Second, we considered the use of 0-B encoding to reduce the number of bitlines that have to be driven during instruction dispatch and issue as well as during forwarding of the results to the waiting instructions in the IQ. Third, we evaluated power reduction achieved by the segmentation of the bitlines within the IQ. Combined, these three mechanisms reduce the power dissipated by the instruction IQ in superscalar processors by 46% to 75% on the average across all SPEC 95 benchmarks depending on the datapath design style and whether spurious tag comparisons are allowed.

The IQ power reductions are achieved with little compromise of the cycle time. Specifically, 0-B encoding logic adds 9% to the cycle time of the datapath with issue-bound operand reads and 12% to the cycle time of the datapath with issue-bound operand reads. The delay of the new comparator roughly doubled compared to the delay of the traditional comparator and therefore the practical use of the proposed comparator is determined by the amount of slack (if any) present in the wakeup stage of the pipeline. If the wakeup cycle determines the processor's cycle time, then the alternative dissipate-on-match comparator designs that we proposed in [20] represent better alternatives. It is very likely that the IQ area increase from the use of all three proposed techniques will not be cumulative. In the worst case, assuming cumulative effect of our techniques on the IQ area, the total area of the IQ increases by about 30%. Our ongoing studies also show that the use of all of the techniques that reduce the IQ power can also be used to achieve reductions of a similar scale in other datapath artifacts that use associative addressing (such as the reorder buffer [23] and load/store queues). As the power dissipated in instruction dispatching, issuing, forwarding and retirement can often be as much as half of the total chip power dissipation, the use of the new comparators, 0-B encoding and bitline segmentation offers substantial promise in reducing the overall power requirements of contemporary superscalar processors.

References

[1] J. Emer. EV8, "The post-ultimate alpha," in *Proc. Keynote Int. Conf. PACT*, Sept. 2001.

[2] D. Folegnani and A. Gonzalez, "Energy-effective issue logic," in *Proc. Int. Symp. Computer Architecture (ISCA)*, 2001, pp. 230–239.

[3] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *Proc. Int. Computer Architecture (ISCA)*, 1997.

[4] K. Ghose, "Reducing energy requirements for instruction issue and dispatch in superscalar microprocessors," in *Proc. Int. Symp. Low-Power Electronics and Design*, 2000, pp. 231–234.

[5] G. Kucuk, K. Ghose, D. Ponomarev, and P. Kogge, "Energy-efficient instruction dispatch buffer design for superscalar processors," in *Proc. Int. Symp. Low-Power Electronics and Design*, 2001, pp. 237–242.

[6] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. HPCA*, 1999, pp. 13–22.

[7] R. Canal, A. Gonzalez, and J. E. Smith, "Very low power pipelines using significance compression," in *Proc. MICRO*, 2000, pp. 181–190.

[8] K. Ghose, D. Ponomarev, G. Kucuk, A. Flinders, P. Kogge, and N. Toomarian, "Exploiting bit-slice inactivities for reducing energy requirements of superscalar processors," in *Proc. Kool Chips Workshop, Held in Conjunction With Microarchitecture*, 2000.

[9] L. Villa, M. Zhang, and K. Asanovic, "Dynamic zero compression for cache energy reduction," in *Proc. MICRO*, 2000, pp. 214–222.

[10] K. Itoh, "Low power memory design," in *Low Power Design Methodologies*, J. Rabaey and M. Pedram, Eds. Norwell, MA: Kluwer, 1996, pp. 201–251.

[11] K. Itoh, K. Sasaki, and Y. Nakagome, "Trends in low-power RAM circuit technologies," *Proc. IEEE*, vol. 83, pp. 524–543, Apr. 1995.

[12] R. J. Evans and P. D. Franzon, "Energy consumption modeling and optimization for SRAMs," *IEEE J. Solid-State Circuits*, vol. 30, pp. 571–579, May 1995.

[13] D. Ponomarev, G. Kucuk, and K. Ghose, "AccuPower: An accurate power estimation tool for superscalar microprocessors," in *Proc. DATE*, 2002, pp. 124–129.

[14] D. C. Burger and T. M. Austin, "The SimpleScalar Tool Set: Version 2.0," Dept. Computer Sci., Univ. Wisconsin, Madison, Tech. Rep., June 1997.

[15] A. Buyuktosunoglu, "An adaptive issue queue for reduced power at high performance," in *Proc. PACS Workshop, Held in Conjunction With ASPLOS*, 2000.

[16] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," in *Proc. MICRO*, 2001, pp. 90–101.

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

[18] D. Brooks *et al.*, "Power-aware micro architecture: Design and modeling challenges for next generation microprocessors," *IEEE MICRO*, vol. 20, pp. 26–44, Nov. 2000.

[19] R. Canal and A. Gonzalez, "A low complexity issue logic," in *Proc. Integrated Computer Solutions (ICS)*, 2000, pp. 327–335.

[20] O. Ergin, K. Ghose, G. Kucuk, and D. Ponomarev, "A circuit-level implementation of fast, energy-efficient comparators for high-performance microprocessors," in *Proc. Int. Conf. Computer Design (ICCD)*, 2002, pp. 118–121.

[21] D. Ernst and T. Austin, "Efficient dynamic scheduling through tag elimination," in *Proc. Int. Symp. Computer Architecture (ISCA)*, 2002, pp. 37–46.

[22] M. Huang, J. Renau, and J. Torrellas, "Energy-efficient hybrid wakeup logic," in *Proc. Int. Symp. Low-Power Electronics and Design (ISLPED)*, 2002, pp. 196–46.

[23] D. Ponomarev, G. Kucuk, and K. Ghose, "Energy-efficient design of the reorder buffer," in *PATMOS'02*, ser. Lecture Notes in Computer Science. Seville, Spain: Springer-Verlag, vol. 2451, pp. 289–299.

[24] G. Lozano and G. Gao, "Exploiting short-lived variables in superscalar processors," in *Proc. Int. Symp. Microarchitecture*, 1995, pp. 292–302.

Dmitry V. Ponomarev (S'96–M'03) received the Systems Engineering degree from Moscow State Institute of Electronics and Mathematics, Russia, in 1996, and the M.S. degree in computer and information science from State University of New York (SUNY), Institute of Technology at Utica/Rome, NY, in 1995, and the Ph.D. degree in computer science from SUNY, Binghamton in 2003. He is currently working toward the Ph.D. degree in computer science at SUNY Binghamton.

He is currently an Assistant Professor in the Department of Computer Science, SUNY Binghamton. His research interests include computer architecture, particularly in the optimizations of high-end microprocessors for energy efficiency.

Gurhan Kucuk (S'00) received the B.S. degree in computer engineering from Marmara University, Istanbul, Turkey, in 1995, and the graduate degree in information and computer science from Yeditepe University, Istanbul, Turkey, in 1999. He is currently working toward the Ph.D. degree in computer science, State University of New York at Binghamton, NY.

His research interests include complexity-effective, energy-efficient microprocessor design.

Oguz Ergin (S'02) received the B.S. degree in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey, in 2000. He is currently working toward the Ph.D. degree in computer science, State University of New York at Binghamton.

His research interests include low-power VLSI design, energy-efficient, and high performance computer architectures.

Kanad Ghose (M'87) received the M.S. and Ph.D. degrees in computer science from Iowa State University, Ames, in 1988 and 1986, respectively.

In 1987, he joined the Department of Computer Science, State University of New York at Binghamton, where he is currently Chair and an Associate Professor. His current research interests inlcude microarchitectural and circuit-level techniques for power reduction, high-performance networking and systems for handling and visualizing large data sets.

Dr. Ghose is a Member of the Association of Computing Machinery (ACM) and of Phi Kappa Phi.

Peter M. Kogge (S'65–M'68–SM'86–F'90) joined IBM, Federal Systems Division, Owego, NY, in 1968, and was appointed an IBM Fellow in 1993. In 1977, he was a Visiting Professor with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, and from 1977 through 1994, he was also an Adjunct Professor with the Department of Computer Science, State University of New York at Binghamton. In August 1994, he joined the University of Notre Dame, Notre Dame, IN, as the first Holder of the endowed McCourtney Chair in Computer Science and Engineering (CSE). Since Summer 1997, he has been a Distinguished Visiting Scientist with the Center for Integrated Space Microsystems at the Jet Propulsion Laboratory, Pasadena, CA. He is also the Research Thrust Leader for Architecture at Notre Dame's Center for Nano Science and Technology. For the 2000–2001 academic year, he was the Schubmehl-Prein Chairman of the Computer Science and Engineering Department at Notre Dame, and since then has been Associate Dean of Engineering for Research.