Scalable Dynamic Instruction Scheduler through Wake-Up Spatial Locality

Chung-Ho Chen, *Member*, *IEEE*, and Kuo-Su Hsiao

Abstract—In a high-performance superscalar processor, the instruction scheduler often comes with poor scalability and high complexity due to the expensive wake-up operation. From detailed simulation-based analyses, we find that 95 percent of the wake-up distances between two dependent instructions are short, in the range of 16 instructions, and 99 percent are in the range of 31 instructions. We apply this wake-up spatial locality to the design of conventional CAM-based and matrix-based wakeup logic, respectively. By limiting the wake-up coverage to $i + 16$ instructions, where $0 \leq i \leq 15$ for 16-entry segments, the proposed wake-up designs confine the wake-up operation to two matrix-based or three CAM-based 16-entry segments no matter how large the issue window size is. The experimental results show that, for an issue window of 128 entries (IW_{128}) or 256 entries (IW_{256}), the proposed CAM-based wake-up locality design saves 65 percent (IW_{128}) and 76 percent (IW_{256}) of the power consumption and reduces 44 percent (IW_{128}) and 78 percent (IW_{256}) in the wake-up latency compared to the conventional CAM-based design with almost no performance loss. For the matrix-based wake-up logic, applying wake-up locality to the design drastically reduces the area cost. Extensive simulation results, including comparisons with previous works, show that the wake-up spatial locality is the key element to achieving scalability for future sophisticated instruction schedulers.

Index Terms—CAM-based wake-up logic, issue logic, low power, matrix-based wake-up logic, scalable instruction scheduler, wake-up spatial locality.

1 Introduction

In order to extract more instruction-level parallelism and boost instructions per cycle (IPC), future generations of high-performance superscalar processors tend toward the specification of using a large issue window with a wide issue width. Consequently, the dynamic scheduler required for out-of-order execution becomes more complicated and less scalable. The complex scheduler consumes a lot of energy and may slow down the clock cycle time of the processor.

In particular, the complexity of the instruction scheduler comes mainly from the wake-up logic that traces the instruction dependences and wakes up the instructions when their source operands become available. The wake-up logic is typically implemented by using the CAM structures that fully match all the source tags in the issue window with the result tags. However, the CAM structures consume a lot of energy and slow down the wake-up speed because of considerable circuit activities and heavy load capacitance. To see this, Fig. 1 shows the IPC and wake-up power dissipation for various issue window sizes of the baseline processor studied in this paper. Table 3 in Section 4 details the simulator configuration. The IPC increases as the instruction window size increases from 32 to 256. As observed, a large window size such as 256 significantly boosts up the IPC; however, to achieve this, the large CAM-based issue window consumes disproportionately much more wake-up power.

The scheduler becomes the major critical path, which limits the clock cycle time, of the pipeline stages, mainly due to the complexity of the CAM-based wake-up logic. Although a pipelined dynamic scheduler can increase the clock frequency, the operations of instruction wake up and instruction selection should be an atomic operation to avoid significant performance degradation. Recent study has shown that the latencies associated with the wake-up and selection form the critical path of the pipeline stages [1], [40]. The wake-up latency increases significantly with both the issue width and window size and the wake-up logic dominates the latency for the scheduler. Increasing the window size and issue width to improve IPC will continue to increase the burden to the clock cycle time.

For energy consideration, the power consumption associated with the scheduler constitutes a significant portion of the processor power consumption. For example, the issue logic is the most power hungry component of the Compaq Alpha 21464 processor, responsible for 46 percent of the total processor power [2], whereas the out-of-order scheduler of the Intel Pentium 4 processor accounts for 40 percent of the total power consumption. As a result, the CAM-based wake-up logic not only slows down the clock speed but also shifts more power budget to the scheduler.

In this paper, we introduce a program metric called wake-up spatial locality, which refers to the fact that the distance between two data-dependent instructions is often short. Based on this observation, two wake-up optimizations that exploit wake-up spatial locality in the CAM-based and matrix-based wake-up logic are proposed. The first optimization divides the monolithic CAM structure into

• *The authors are with the Department of Electrical Engineering, National Cheng Kung University, No. 1 Ta-Hsueh Road, 701 Tainan, Taiwan. E-mail: chchen@mail.ncku.edu.tw, newjimmy@ee.ncku.edu.tw.*

Manuscript received 28 June 2006; revised 8 Dec. 2006; accepted 15 May 2007; published online 6 June 2007.
Recommended for acceptance by A. González.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0254-0606.
Digital Object Identifier no. 10.1109/TC070743.

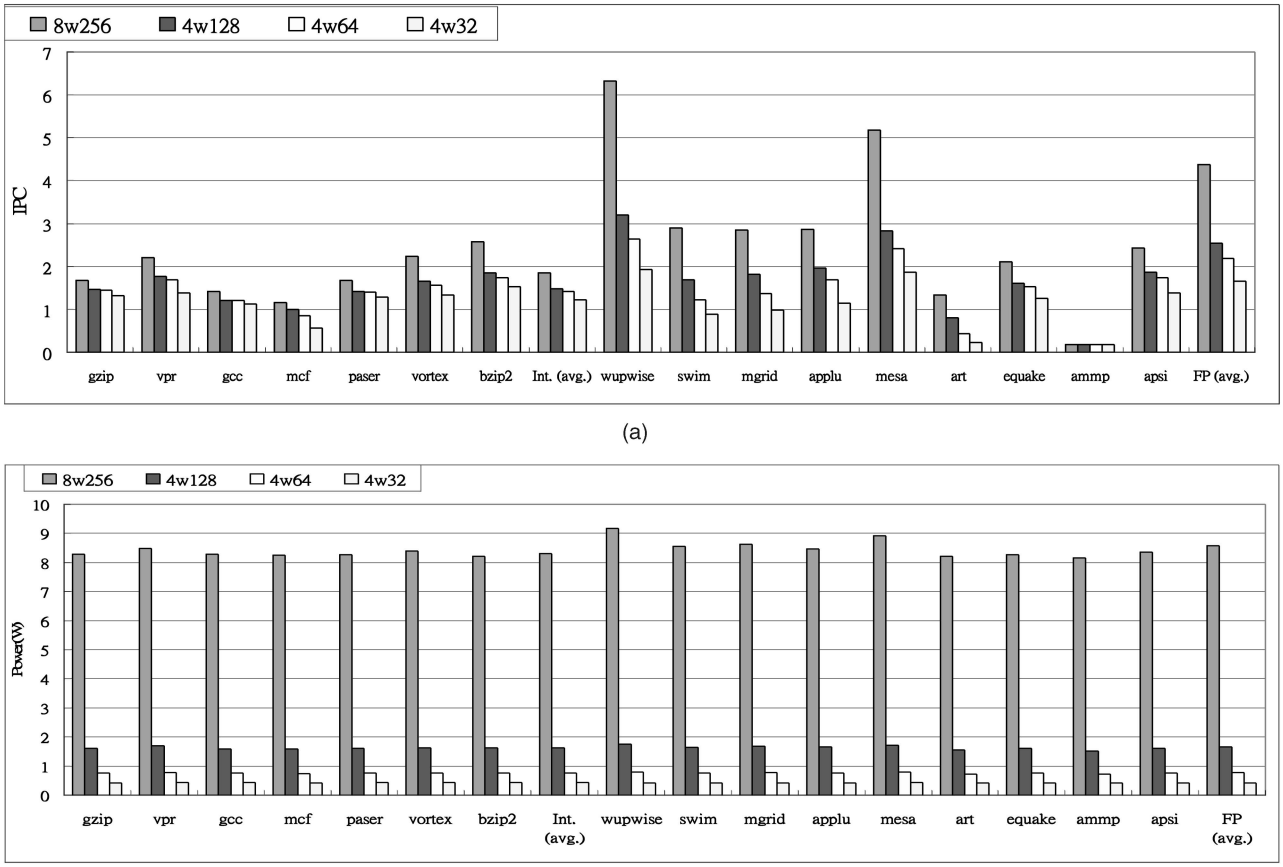

Fig. 1. IPC and power consumption of wakeup operation for various issue window size. (a) IPC for 4-wide 32, 64, 128-entry, and 8-wide 256-entry issue window. (b) Power consumption for 4-wide 32, 64, 128-entry, and 8-wide 256-entry issue window.

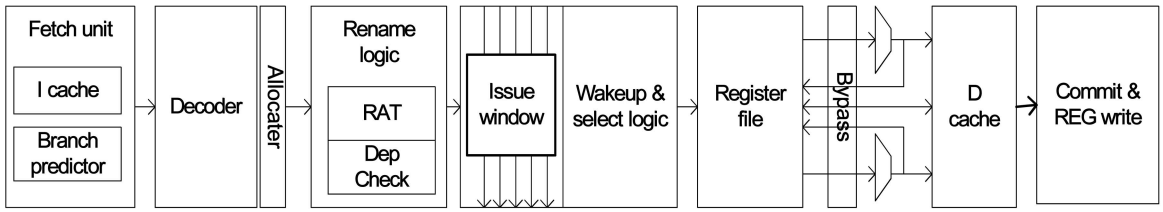

Fig. 2. Processor model used in this paper.

multiple segments and activates only the necessary segments during the wake-up process. This design greatly reduces the power consumption and wake-up latency of the associative lookup operations. The other optimization divides the monolithic wake-up logic matrix into multiple segments and thus narrows the segment width to reduce the area cost of a conventional matrix design. Both of the proposed wake-up designs achieve excellent scalability because the number of segments which need to be activated during each wake-up process does not increase with the issue window size.

The remainder of this paper is organized as follows: Section 2 discusses the baseline processor model used in this paper, explains the limitations of the wake-up logic used in dynamic schedulers, and provides a brief review of related works. Section 3 presents the wake-up spatial locality found in representative benchmark programs and details the proposed wake-up designs. Section 4 presents the experimental methodology and the evaluation results. In Section 5, we discuss the area usage and variant designs. Finally, Section 6 concludes this paper.

2 Background

2.1 Baseline Processor Model

Fig. 2 depicts the baseline superscalar processor model used in this paper. The fetch unit retrieves multiple instructions from the instruction cache with a branch predictor to speculatively fetch instructions over basic blocks during a clock cycle time. Subsequently, the instructions are decoded and their register designators are renamed for resolving the WAR and WAW dependences. Then, the instructions are dynamically scheduled for out-of-order execution. After scheduling, their source operands are read from the register

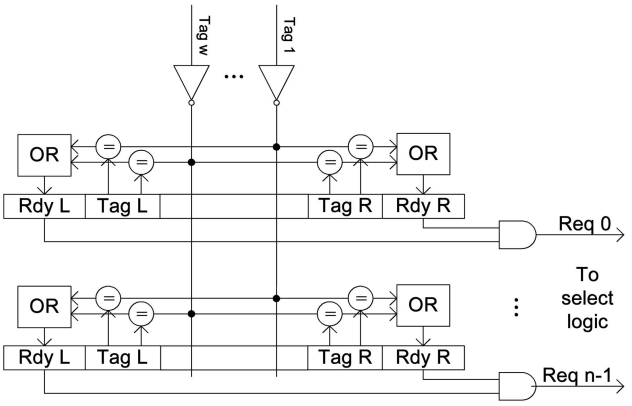

Fig. 3. Wake-up logic implemented by using the CAM structure.

file or bypassed from the functional units. Finally, the instructions are committed in the program order to ensure the correct completion of the program. This scheduling model is implemented in HP PA8000 [3], Intel Pentium 4 [4], MIPS R10000 [5], Alpha 21264 [6], and its successors.

In the scheduler stage, the wake-up and select logic directs the instructions that are waiting for their source operands to become available or waiting for execution. After rename, the instructions are inserted into the issue window to wait for their operands or to wait for execution. The ready instructions send signals to the select logic to request execution. Once a functional unit becomes available, the select logic directs a suitable instruction to that unit for execution by asserting the corresponding grant signal. Many selection policies, for instance, the oldest first selection algorithm [7], have been presented for the case where the number of ready instructions exceeds the capacity of the available functional units [7], [8].

In the baseline processor, the grant signals from the select logic are used not only to select instructions for execution but also to wake up the dependent instructions. When forwarding the grant lines for the wake-up operations, the grant lines are delayed according to the execution cycle time of the corresponding instructions. Only the grant lines with one cycle execution time are immediately used for the following wake-up operation. To wake up the dependent instructions, the asserted grant lines are used as the index addresses to read their corresponding destination tags. Then, the destination tags are forwarded to the wake-up logic to match with the entire source tags in the issue window.

Another processor model uses a reorder buffer and reservation stations for dynamic instruction scheduling, such as the model in the Intel P6 [9], PowerPC 604 [10], and HAL SPARC64 [11]. For this model, the register read stage is placed before the scheduler stage. After being renamed, the available operands are read from the register file or the reorder buffer and then inserted into the reservation station, together with the corresponding source tags. After execution, both the result tags and result values are forwarded to the wake-up logic in the reservation stations to wake up the dependent instructions. More discussions on this scheduling model can be found in [12].

2.2 Limitations of Conventional Wake-Up Logic

Fig. 3 shows the conventional implementation of the wake-up logic based on the CAM structure [1]. An extensive survey for CAM circuits and architectures can be found in [45]. In Fig. 3, the wake-up design employs two CAM structures to match the result tags with the left and right source tags. Two ready bits (Rdy L and Rdy R) are employed for each entry to indicate whether their corresponding operands are available or not. For the wake-up operation, the result tags are driven on the tag buses (Tag 1 to Tag w) into the CAM structures to match with the left and right source tags (Tag L and Tag R). If one of the result tags is matched with the source tag, the corresponding ready bit is set to indicate that this operand is available.

The nature of the CAM structure is inefficient in terms of energy usage and latency. During each wake-up process, many tag lines should be driven and the load capacitance on each tag line is heavy for driving all match circuits of the CAM. Additionally, many match lines are activated in the wake-up operation whether it is a match or not. Both the tag driving and match activities consume a lot of energy and slow down the wake-up speed.

In an effort to improve IPC, scheduler designs often employ a larger window and more aggressive issue width. In other words, a larger window will lead to heavier load capacitances and more match activities; wider issue width means that more tag lines need to be driven. We can see that increasing the window size and issue width leads to larger power consumption and slower wake-up speed. As a result, the scheduler cannot scale well with the increase in window size and issue width.

In addition to the CAM-based design, an alternative approach is implemented by using the bit matrix structure [13], [14]. Fig. 4 shows this matrix-based wake-up design. This wake-up design employs two bit-map memory

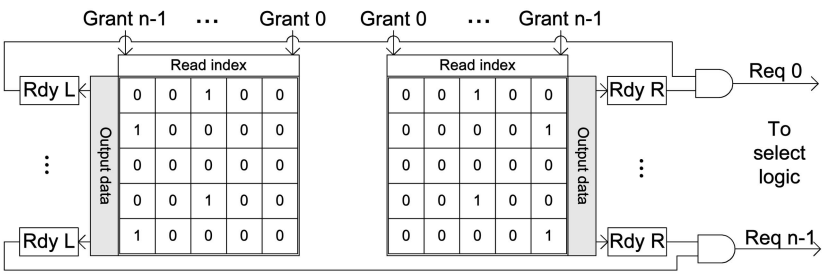

Fig. 4. Wake-up logic implemented by using the bit-matrix RAM.

structures, of which the height and width are both identical to the issue window size, to handle the wake-up operations. In the bit matrix, each bit position represents the data dependence between two instructions. For example, the bit located at the intersection of the ith column and the jth row in the matrix indicates that the jth instruction requires a source operand from the output of the ith instruction. In the same way, other instructions that depend on instruction i will set the corresponding bits in column i.

Differently from the structure of a general RAM, which outputs a row of data, this matrix outputs a column of data. To wake up the dependent instructions, the grant lines read the corresponding columns from the matrix. If the bits on the selected column have been set, the corresponding ready bits are set to indicate that the result is now available for the waiting instructions. This RAM-based wake-up design has advantages in power dissipation and latency; however, it is not scalable due to its prohibitive area requirement for a large issue window. The wire delay of the wake-up path increases as the logic size grows, leading to excessive wire delay in the processes of future technology [15].

2.3 Related Work

Numerous previous efforts have attempted to improve the performance of the CAM-based scheduler, aiming at reducing the power dissipation, wake-up latency, or both. The bank design in [1], which is most similar to our design, segments the monolithic issue window into multiple banks to improve wake-up delay; however, the result tags still need to be broadcast to all the banks. This design induces extra wake-up delay and power consumption due to the additional driver transistors and tag buses. Hrishikesh et al. proposed a pipelined-wake-up design that segments the issue window and wakes up the instructions in the segments in multiple sequential cycles [16]. Nonetheless, the limitations are that all of the segments still need to be searched and the dependent instructions can be issued back to back only if they are in the first segment.

Folegnani and González presented a gate-off technique that disables the useless (empty and ready) entries of the issue window from tag matching [17]. Ramirez et al. proposed a similar gate-off mechanism based on a multibank issue window to improve the power consumption of the scheduler [18]. The mechanism has employed an extra large RAM structure that may slow down the wake-up speed. Several approaches dynamically manage the sizes of the issue window and turn off the useless entries [17], [21], [22], [23]. These designs improve the power consumption of the scheduler with extra dynamic managers that may complicate the scheduler.

A tag-elimination scheduler that employs fewer tag comparators to reduce the complexity of the scheduler has been proposed by Ernst and Austin [24]. This scheduler also has a last tag speculator to reduce the frequency of tag matching. Based on the same observation, Sharkey et al. presented an instruction-packing technique. This technique schedules the two instructions, which have only one nonavailable source operand, into the same entry of the issue window [25].

Kim and Lipasti proposed a sequential wake-up mechanism to reduce the complexity of the scheduler [26]. This mechanism places the last-arrival operand into the fast wake-up logic and wakes up the left and right source operands of an instruction in two sequential steps. Aggarwal et al. proposed a reduced wake-up width scheduler to reduce the complexity of the scheduler by reducing the maximum number of input result tags for the wake-up logic [27].

The wake-up-free schedulers [28], [29] predict the issue latency of the instructions and then issue the instructions into a FIFO-based issue queue. These schedulers replace the complex wake-up logic with a simple FIFO queue. On the other hand, Brown et al. presented the select-free scheduler [13]. This scheduler removes the instruction-selection process from the scheduling critical path.

Several proposals [30], [31], [32] employ a two-level issue window to reduce the complexity of the scheduler. The critical instructions are dispatched to the small and first issue window and the noncritical instructions, for example, the instruction waiting for a load that misses in cache, are dispatched to the large and slow window.

On the other hand, many wake-up designs employ custom components instead of CAM structures. Goshima et al. presented a wake-up design that uses bit matrix structures [14]. Henry et al. presented a cyclic segmented prefix (CSP) circuit to improve the performance of the wake-up logic [33]. Hsiao and Chen presented a wake-up design which predecodes the source tags and matches the decoded outputs directly with the grant lines to improve the wake-up speed and power consumption [34]. Ponomarev et al. used three techniques, efficient comparators, 0-B encoding, and bitline segmentation, to reduce the energy dissipation of the issue window [35].

Several designs reduce the complexity of the issue logic through index-based techniques using pointers to connect the producer instructions and consumer instructions [36], [37], [38]. Some works reduce the complexity of the scheduler by prescheduling dependent instructions into a data-flow-based issue window [39], [40], [41].

3 Scalable Wake-Up Logic through Wake-Up Spatial Locality

In this section, we first show that most of the distances between two data-dependent instructions are short in programs. Motivated by this wake-up spatial locality, two scalable wake-up designs, a CAM-based one and a matrix-based one, are proposed for the dynamic scheduler.

3.1 Wake-Up Spatial Locality

The wake-up spatial locality is an inherent spatial feature of two data-dependent instructions in programs, where the consumer instruction is often close to the producer instruction. We call this program characteristic "wake-up spatial locality" or simply "wake-up locality" for the wake-up operation in an out-of-order execution processor. The wake-up spatial locality is measured by the instruction count between two data-dependent instructions in the program. The instruction count is also referred to as the wake-up distance.

To quantify the degree of the distance between two dependent instructions, a dynamic scheduled processor

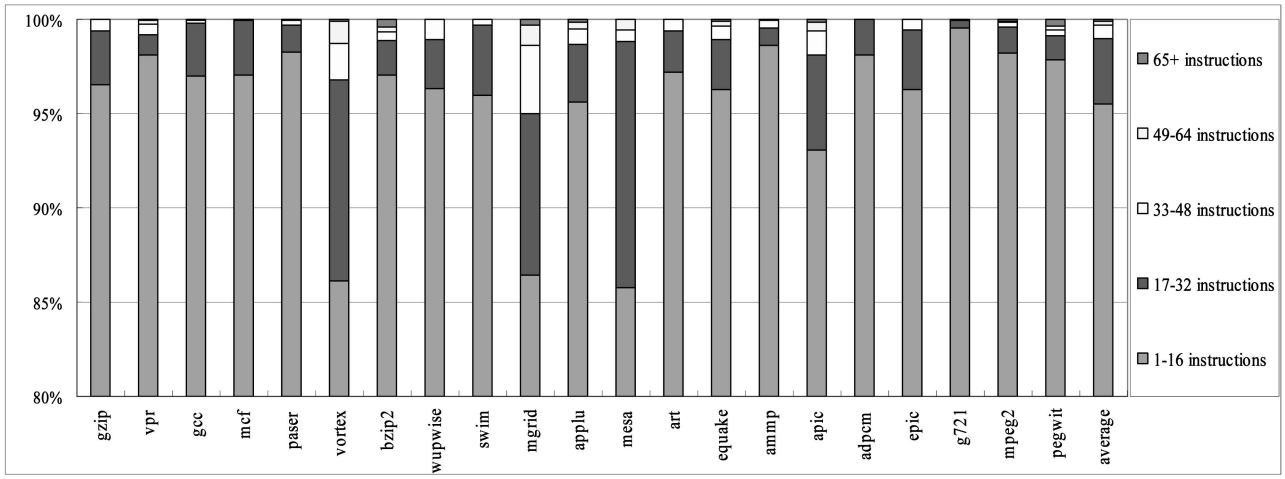

Fig. 5. Runtime distribution of the wake-up distances for a 4-issue 128-entry processor.

was simulated. When instructions were dispatched into the issue window, the instruction distance between two dependent instructions was counted. Fig. 5 shows the distribution of the wake-up distances for all of the wake-up operations of the simulated programs. The statistics are based on a 4-wide processor with a 128-entry issue window (the simulation environment for the experiment is presented a later section). Results are shown for 16 of the SPEC2000 benchmarks and five of the Media-bench programs [44].

As observed, 95 percent of the wake-up distances are within the range of 16 instructions on the average. Moreover, only about 1 percent of the wake-up operations come with the wake-up distances larger than 32 instructions. In particular, applications such as the gcc, mcf, adpcm, and g721 programs show a very strong wake-up spatial locality. Almost all of the wake-up operations have wake-up distances of less than 32 instructions.

The wake-up spatial locality comes from the fact that dependent instructions are often arranged in proximity to improve the data transfer and register utilization. For the integer and media benchmark programs, the wake-up distances are shorter than those of the floating-point programs. This is because the integer and media programs tend to have less global value passing and smaller subroutines.

3.2 Scalable CAM-Based Wake-Up Logic

It is possible to take advantage of the wake-up spatial locality by searching fewer entries of the CAM-based wake-up logic during the wake-up process. Conventional wake-up design employs monolithic CAM structures to handle wake-up operations with wake-up distances up to the issue window size. Due to the heavy load capacitance and considerable circuit activities, this monolithic CAM design has poor scalability and is inefficient both in terms of energy and speed. Since the distances of wake-up operations are often short, it is not necessary to search all of the source tags in the wake-up logic during the wake-up process. The proposed design takes advantage of this locality by using smaller segmented CAM structures that support shorter wake-up distance.

The smaller CAM segments used in this design are classified into two types, a full segment (Fseg) and multiple reduced segments. The reduced segments support the wake-up operations only for instructions having wake-up distances in the limited wake-up range. The Fseg is used to handle wake-up operations for instructions that are out of the limited wake-up range.

Recalling that 99 percent of the wake-up distances are less than or equal to 31 instructions and 95 percent less than or equal to 16 instructions, this indicates that a good segment size to use is 16. In order to cover up to 31 instructions for the wake-up distance, assignment of instructions to the reduced segments can be arranged, as shown in Fig. 6, where the logical structure of the 256-entry issue window is assumed to be a circular ring. Specifically, the wake-up range of an instruction at position i in a reduced segment can be described as $i + \mathbf{16}$, where $0 \leq i \leq 15$. That is, the instruction at location i of a reduced segment can be awakened by a previous source instruction that is as far as $i + 16$ instructions away. In this way, the wake-up range for the instruction at the first entry of a reduced segment includes the 16 preceding instructions before the first entry and, consequently, the wake-up range for the instruction at the last entry of a reduced segment includes the 31 preceding instructions ahead of the last entry.

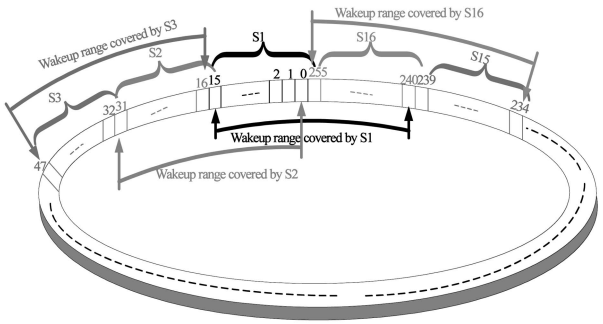

Fig. 6. Instruction assignment and wake-up coverage for the reduced segments $(\mathrm{IW} = 256)$.

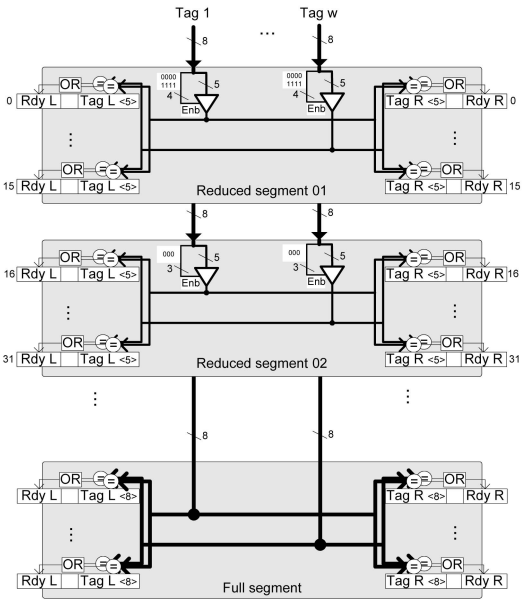

Fig. 7. An example of the CAM-based wake-up locality design for a 256-entry issue window.

Fig. 7 illustrates the example of the proposed wake-up design for a 256-entry issue window. A total of 16 reduced segments and one Fseg are used. Each reduced segment consists of 16 entries of the CAM structure. Each entry is assigned an entry number in the same sequential order as that in the conventional design.

Since the wake-up range of the reduced segments is limited to the coverage of $\boldsymbol{i}+\mathbf{16}$ instructions, where $0 \leq \boldsymbol{i} \leq 15$, the bit length of the inputs (result tags) and the source tags for matching can be reduced to five bits instead of $\log_2$ IW, where IW is the issue window size. For the example in Fig. 7, the least significant five bits of an 8-bit result tag are used as inputs for the reduced segments and, thus, the source tag fields in the reduced segment store only the five low-order bits of the source tags.

In addition to the reduced segments, an Fseg, shown at the bottom in Fig. 7, is employed to handle the wake-up operations for the instructions with the wake-up distances out of the range supported by the reduced segments. The Fseg is a small segment of the conventional wake-up logic that handles the wake-up operations without constraint on the wake-up range. Since only about 1 percent to 5 percent of the wake-up distances of the dynamic instructions are out of the limited range, an Fseg of 16-entry can handle the wake-up operations for the out-of-range instructions well.

		IW # (dest. tag)	Assigned segments	Source Tag (L)	(R)
I_0	R_1 , Rx , Ry	000 00000	S1	11100	11101
⋮		⋮			
I_3	R_4 , Rx , Ry	000 00011	S1	11100	11101
⋮		⋮			
I_{28}	R_6 , R_1 , R_4	000 11100	S2	00000	00011
⋮		⋮			
I_{34}	Rx , R_1 , R_6	001 00010	Full	00000000	00011100

Fig. 8. Illustration of source tag insertion in the split segments.

More discussion about the trade-off between the performance and the entry number used by the Fseg will be presented later. One side issue arising due to the use of the Fseg is that the instruction order is no longer maintained as in the case of a single issue window. Despite this, they are still in their respective age order in the reduced segments, as well as the Fseg, allowing the select logic to do an age-based issue policy. We will revisit this subject in the performance section.

The insertion of a source tag into the proposed wake-up design works as follows: After rename, the instruction is allocated a destination tag (entry number) that indexes the issue window for writing into. This destination tag is also used to select a reduced segment and allocate an entry from the selected segment for writing the source tag into, provided that both of the source tags of the instruction are within the specified wake-up distance. On the other hand, the source tag of the instruction with a wake-up distance out of the range of the allocated segment is inserted into the Fseg, including that one source tag is within the range, whereas the other is not, of the same instruction. As illustrated in Fig. 8, the source tags of instruction I_{28} are allocated to the reduced segment S2 since both of the source instructions that generate operands, R_1 and R_4, are within the specified wake-up coverage. However, the source tags of instruction I_{34} must be allocated to the Fseg because one of its source operand generating instructions, I_0, is out of the specified range. Note that, if the distance of I_0 were within the coverage, then I_{34} would be allocated to the reduced segment S3. In this example, it is assumed that instruction I_0 has long execution latency and I_{34} must wait for the wake up, whereas both registers R_x and R_y are within the specified range with a destination tag of 11111100 and 11111101, respectively.

It is straightforward to determine whether the wake-up distance is out of range or not by checking the most significant bits (distance codes) of the source tag. Table 1

TABLE 1
Distance Codes and the Corresponding Wake-Up Coverage for a 256-Entry Issue Window

Segment	S 1	S 2	S 3	S 4	…	S 16
Distance codes	0000, 1111	0000,0001 or 000x	0001, 0010	0010,0011 or 001x	…	1110,1111 or 111x
Wakeup coverage	0-15, 240-255	0-31	16-47	32-63	…	224-255

TABLE 2
Distance Codes and the Corresponding Wake-Up Coverage for a 128-Entry Issue Window

Segment	S 1	S 2	S 3	S 4	…	S 8
Distance codes	000, 111	000,001 or 00x	001, 010	010,011 or 01x	…	110,111 or 11x
Wakeup coverage	0-15, 112-127	0-31	16-47	32-63	…	96-127

shows the distance codes and the wake-up coverage of the respective reduced segment. The distance codes are simply the segment identifiers, which are numbered from 0 to IW/SS-1, where IW is the issue window size and SS is the segment size. The number of bits used by the distance code equals $\log_2$ (IW/SS).

To compare the wake-up coverage, the most significant four bits of the two source tags are matched with the distance code(s) shown in Table 1 according to the allocated segment. For the even segments, S2 for example, we do not care about the last bit of the codes so that a single three-bit code is enough. If there is a match, the wake-up distance of the source tags is in the wake-up range of the allocated segment and, then, the least significant five bits of each source tag are inserted into the allocated entry. If the wake-up distance is out of the range, then the two source tags are inserted into the Fseg. In the above example, the most significant three bits (000) of the two source tags for I_{28} are used to compare with the distance code of S2 in Table 1, respectively. The match indicates that the source tags of I_{28} should be allocated to S2. As for I_{34}, the most significant four source tag bits for R_1 is 0000, which is compared with the distance code for S3 according to the IW entry number of I_{34}. In this case, it is a mismatch and, thus, the source tags of I_{34} should go to the Fseg.

During the wake-up process, the result tag is always used as the input for the Fseg; however, the result tag is only used as the input for reduced segments that have the same distance codes as the result tag. As a whole, only two reduced segments and the Fseg are activated for matching. Continuing with the example in Fig. 8, in the wake-up process, assume that I_0 has completed the execution and its result tag, 00000000, is driven. Only S1 and S2 out of the reduced segments are searched because of the same distance code 0000, whereas the Fseg is always searched.

For comparison, Table 2 shows the distance codes and the wake-up coverage for a 128-entry issue window. The source tag field remains 5-bits long since the wake-up coverage is still $\boldsymbol{i + 16}$ instructions for $0 \leq i \leq 15$, whereas the distance codes need only three bits $(\log_2 128/16 = 3)$.

Compared to the conventional design, the proposed design has three major advantages: smaller load capacitance on the tag bus, shorter length of the source tag fields in the reduced segments, and fewer match activities during the wake-up process. These factors significantly improve the power consumption and wake-up latency of the scheduler that employs a relatively large issue window. Another advantage of this design is the excellent scalability. No matter how large the issue window size is, the number of the activated segments remains the same during the wake-up process.

3.3 Scalable Matrix-Based Wake-Up Logic

In this section, we present the application of wake-up spatial locality in a matrix-based wake-up design. The bit matrix is an alternative way to implement the associative match operations in the wake-up logic. A wake-up logic that employs bit matrices is efficient in terms of latency and energy usage. However, this design comes at the expense of a large area cost when the issue window size is increased because the area of this design is proportional to the square of the issue window size. Based on the wake-up spatial locality, the area requirement can be greatly reduced by limiting the wake-up range of the matrix-based design. Similarly to the CAM-based wake-up locality design, the proposed optimization divides the monolithic bit matrix into multiple segments. The basic idea of this design is to reduce the area cost by narrowing the width of the matrix structures.

The structure of the matrix-based wake-up locality design is shown in Fig. 9. There are three major differences between this design and the CAM-based wake-up locality design. First, the components used in this design are the RAM-like structures (matrixes) not the CAM structures. Second, the inputs for this design are the grant lines from the select logic.

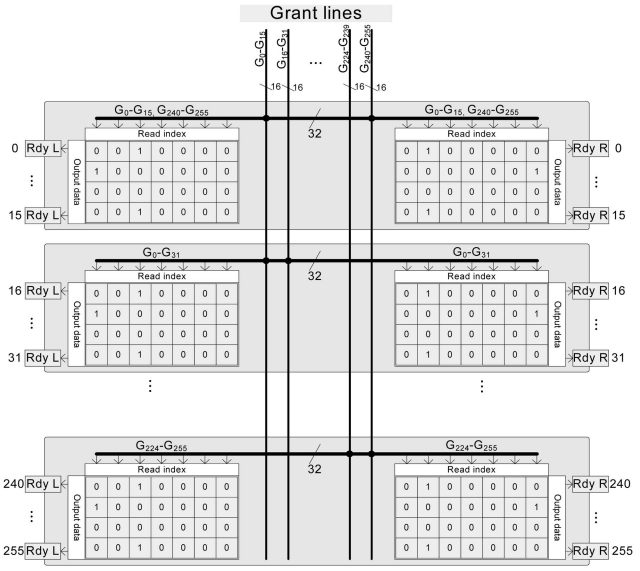

Fig. 9. An example of the matrix-based wake-up locality design for a 256-entry issue window.

Last, there is no Fseg for this design. The segments are small bit matrices; the width (input lines) of the matrices is reduced by limiting the wake-up range of each segment. Similarly, the input lines (width) of a 16-entry segment can be reduced to 32 grant lines, where the wake-up range of this segment is limited to $i + 16$ instructions, where $0 \leq i \leq 15$. The 32 grant lines for each segment are arranged as the wake-up coverage shown in Table 1 for a 256-entry issue window. The wake-up coverage can be extended by having the segments connected with more grant input lines, for instance, 48 grant lines for the wake-up coverage of $i + 32$ instructions $(0 \leq i \leq 15)$. The results of a bit-matrix design with different wake-up coverage are presented later.

In the bit matrix, each bit position represents the data dependence between two instructions. For example, the bit located at the intersection of the ith column and the jth row in the bit matrix indicates that the jth instruction requires a source operand from the output of the ith instruction. In the same way, other instructions that depend on instruction i will set the corresponding bits in column i.

An instruction's source tag can only be inserted into the allocated segment when the wake-up distance of this instruction is within the wake-up range. The wake-up distance of the instruction is checked using the distance codes, as mentioned in the previous section. If it is in the range of the allocated segment, the destination tag of the instruction is used as an address to select a row and the five low-order bits of the source tag are decoded to select a column and the bit at the intersection is set. In other words, only one bit is set in the selected row during each insertion. If the wake-up distance is out of the wake-up range, the insertion of instruction is stalled until this dependence is resolved.

Since the bit matrix is a RAM-like structure, the wake-up operation of this design can be finished through a read from the bit matrix. The difference between the bit matrix used here and a general RAM structure is that the bit matrix outputs a column of data, whereas a general RAM structure outputs a row of data. The wake-up operation is performed through a column read from the selected segment. The asserted grant lines read the corresponding columns from the bit matrix. If the bits on these selected columns have been set, the outputs of these bits drive the corresponding ready bits to indicate the availability of the operands.

Differently from the CAM-based wake-up locality design, this design employs no Fseg for handling the out-of-range instructions. This is because the matrix-based Fseg is inefficient in terms of area usage for the number of grant lines (256, for instance) it must connect to. Besides, an Fseg of this kind slows down the wake-up speed since the wake-up delay of the Fseg is longer than that of the reduced segments.

This matrix-based wake-up locality design removes the fatal disadvantage of the conventional bit matrix scheme by greatly reducing the area cost. The design with 16-entry and 32-bit matrixes takes only 12.5 percent (IW_{256}) and 25 percent (IW_{128}) of the area cost of the conventional matrix design. This design is much faster and more energy-efficient because only two segments, which are much smaller than that of the monolithic matrix, are activated during the wake-up process. The downside of this design is that limiting the wake-up range may block the insertion of instructions into the issue window.

4 Experimental Evaluation and Analysis

This section presents the experimental methodology and discusses the results of latency, power, and performance for the proposed optimizations and previous designs.

4.1 Experimental Methodology

The power consumption and IPC results of the evaluated designs were obtained through architectural simulation, which was conducted by using the Wattch [42] and SimpleScalar [43] toolsets. These execution-driven simulators simulate a superscalar processor with two-level caches, branch predictors, dynamic scheduler, and so forth, by performing cycle-by-cycle instruction-level simulation, including execution down any speculative path, until a branch misprediction is detected.

Table 3 lists the architectural parameters for the 4-wide and 8-wide superscalar processors. In Wattch, the RAM cell of the bit matrix was extended from the conventional RAM cell. The other configurations for the Wattch include a 1 GHz clock frequency, 1.8 V voltage, and 0.18 μm technology process. The simulation results were collected from seven integer and nine floating-point programs of the SPEC2000 benchmark suite. The test input set was used for the SPEC2000 benchmark programs. Additionally, five programs of Media-bench were also employed for a more comprehensive evaluation. All of the selected benchmark programs were compiled with full optimization (-O4). The programs were fastforwarded for the first 50 million instructions and the following 500 million instructions were simulated.

To understand the effects on the wake-up delay, the circuit characteristics of the evaluated designs must be examined. The circuit models were extended from the one proposed by Ernst and Austin [24] and the timing results for the evaluated designs were extracted by using the Avant! Hspice tool. Finally, the parameters of CMOS transistors and wires were all conformed to the TSMC 0.18 μm technology process.

4.2 Performance Comparisons

Figs. 10 and 11 present the IPCs of the 4-wide and 8-wide processors that employ different wake-up logic. These results are normalized to the IPC of the baseline processor that employs the conventional CAM-based wake-up logic. Besides, the IPC performance of the bank design [1], the conventional matrix design, and the gated-off design [17] are the same as that of the baseline processor. The bank design segments the monolithic CAM structure into multiple banks. The bit matrix design replaces the CAM structures with the RAM-like structures. The gated-off design gates off the ready and empty entries of the CAM structures. All three of these designs do not change the architectural configurations; thus, the IPC results are the same as that of the baseline processor.

The first two bars show that the IPC drops due to the tag elimination design and sequential wake-up design. The tag

TABLE 3
Processor Configurations

	4-wide	8-wide
Out-of-order Execution	4-wide fetch/issue/commit, 128 RUU, 64 LSQ.	8-wide fetch/issue/commit, 256 RUU, 128 LSQ.
Functional units	4 IALU, 1 IMUL, 2 FALU, 1 FMUL, 2 LSU.	8 IALU, 2 IMUL, 4 FALU, 2 FMUL, 4 LSU.
L1 I-cache L1 D-cache	4-way, 64KB, 32-byte line, 2-cycle latency. 4-way, 64KB, 32-byte line, 2-cycle latency.	
L2 cache TLB	4-way, 512KB, 64-byte line, 10-cycle latency. 4-way, 128-entry, 4KB page size.	
Memory	64-bit wide, 75-cycle latency, 4-cycle burst.	
Branch predictor	Combination of bimodal (2k entries) and 2-level global predictor (2k entries, 8-bit history), 1024-entry chooser, 1024-entry (4-way) BTB, 16-entry RAS (return address stack), 8-cycle penalty.	

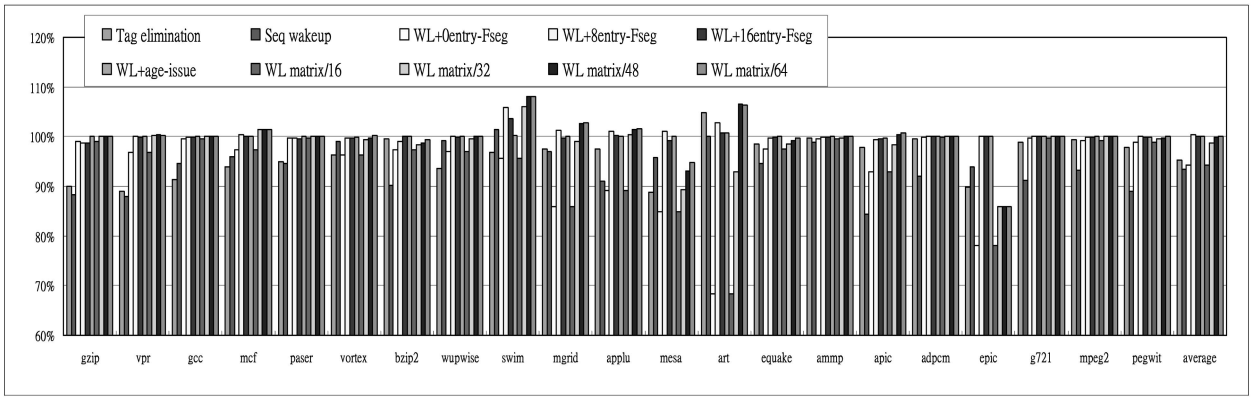

Fig. 10. The normalized performance of the 4-wide 128-entry processor with a different wake-up logic.

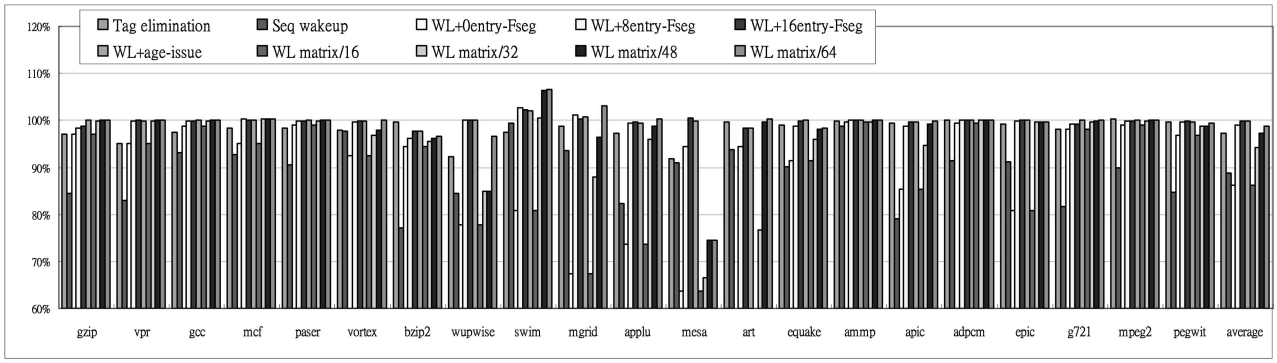

Fig. 11. The normalized performance of the 8-wide 256-entry processor with a different wake-up logic.

elimination design, configured as 32 two-tag stations, 64 one-tag stations, and 32 zero-tag stations for the 4-wide processor and twice the stations for the 8-wide processor, loses about 3 percent (IW_{256}) and 5 percent (IW_{128}) of IPC due to the issue policy and the capacity conflicts. Since the number of stations is more sufficient in the 8-wide configuration, the capacity conflicts do not occur frequently and this slightly mitigates the IPC loss. On the other hand, the IPC drop due to the sequential wake-up design is measured to be 7 percent (IW_{128}) and 11 percent (IW_{256}). Obviously, waking instructions up in two sequential cycles induces nonnegligible performance degradation.

There is almost no performance degradation for the proposed CAM-based wake-up locality (WL + 16-entry-Fseg) design. The IPC loss is measured to be only 0.2 percent for the 8-wide processor and no IPC loss for the 4-wide processor as shown in the fifth bars. The slight IPC drop for

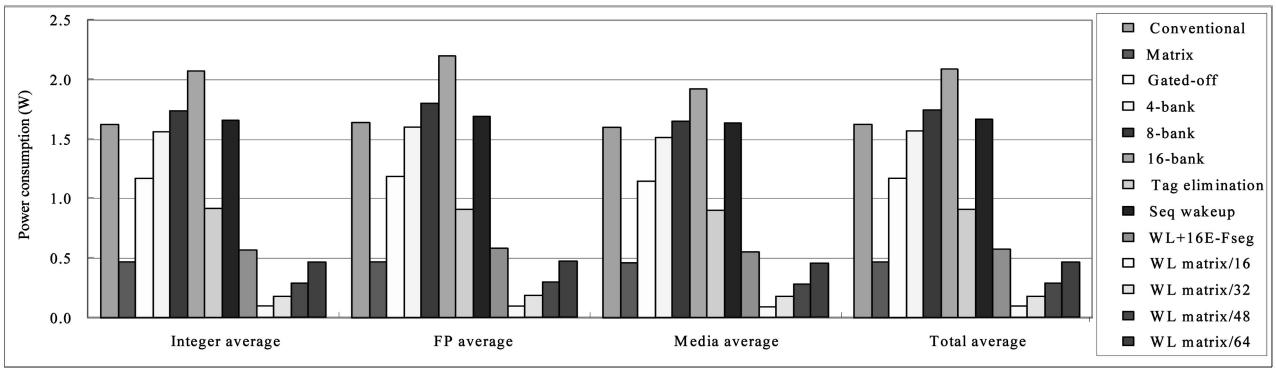

Fig. 12. Power consumption of the wake-up designs for a 4-issue 128-entry processor.

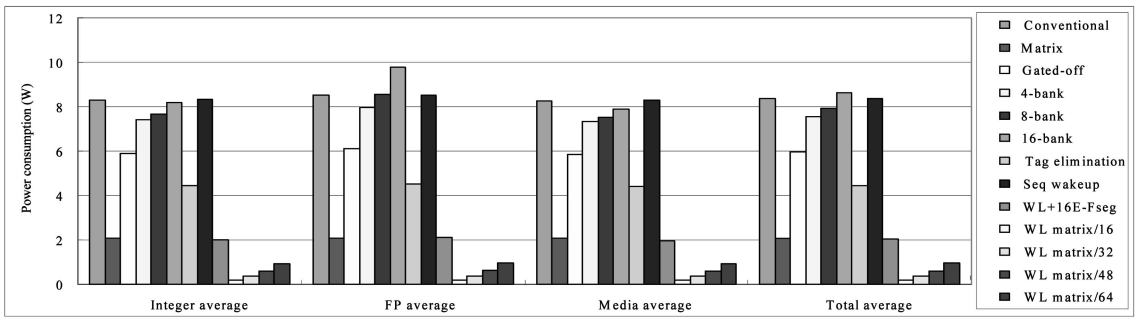

Fig. 13. Power consumption of the wake-up designs for an 8-issue 256-entry processor.

the 8-wide processor comes from the extra dispatch stall when the Fseg has no more available entry for the incoming out-of-range instruction.

The third bars show the performance for the proposed design without the Fseg. Although only 1 percent to 4 percent of the dynamic instructions are out of the wake-up range, the dispatch stalls due to these instructions induce 6 percent (IW_{128}) and 14 percent (IW_{256}) IPC drop. The fourth bars show that using an 8-entry Fseg can avoid most of the performance degradation due to the dispatch stalls. In addition, the sixth bars show the performance of the wake-up spatial locality design with the oldest first issue policy. The performance is the same as that of the WL design without the oldest-first issue policy. In the wake-up locality design, since the instructions are dispatched into the reduced and Fsegs, the instruction order cannot be kept the same as that in the conventional design. During the instruction issue process, the proposed design cannot do the oldest first issue. Instead, the instructions in the reduced segments are assigned a higher priority than those in the Fseg; we call this policy dual-priority aged issue. Although the instructions lose their program order, they are still in their age order in the reduced segments and Fseg, respectively. Based on the position (entry number), the instructions in the reduced segments or the Fseg can be issued in the aged-issue policy. The performance of the WL design with the dual-priority aged-issue policy, shown in the fifth bars, is as good as that of the WL design with the oldest first issue policy, shown in the sixth bars.

Finally, the last four bars show the performance of the matrix-based wake-up spatial locality design (WL matrix) with different configurations. For the configuration that the wake-up range is limited to $\boldsymbol{i+16}$ instructions (WL_matrix/$\mathbf{16}$), although only 4 percent of the dynamic instructions are out of the wake-up range, the dispatch stalls due to these instructions induce 6 percent (IW_{128}) and 14 percent (IW_{256}) IPC drop. As expected, the IPC of this design becomes better when the wake-up range increases. The performance degradation is close to 0 percent (IW_{128}) and 1.3 percent (IW_{256}) after increasing the wake-up coverage to $\boldsymbol{i+64}$ instructions (WL_matrix/$\mathbf{64}$), where $0 \leq \boldsymbol{i} \leq 15$.

To summarize, although the wake-up range is limited to $\boldsymbol{i+16}$ instructions $(0 \leq \boldsymbol{i} \leq 15)$, the CAM-based wake-up locality design achieves almost no performance degradation due to the employment of the Fseg. As for the matrix-based wake-up locality design, the performance impact can be reduced to 1.3 percent when the wake-up coverage grows to $\boldsymbol{i+64}$ instructions. For some of the cases where the normalized IPC is greater than 100 percent, we examined these simulation scenarios and found the cause coming from branch prediction. With the wake-up range being limited, fewer consecutive branches will be issued and this reduces the chance of misprediction in these circumstances.

4.3 Power Consumption

Figs. 12 and 13 present the power consumption of the wake-up logic for the 4-wide and 8-wide processors. Note that, in this paper, only dynamic power is evaluated. The power consumption of the conventional CAM design, shown in the leftmost bars, is found to be much higher than others. This is due to the heavy load capacitance and the surplus circuit activities of the monolithic CAM structure. The gated-off design reduces 28 percent (IW_{128}) and 29 percent (IW_{256}) power consumption of the CAM scheme by gating the ready and empty entries from tag matching. Due to the

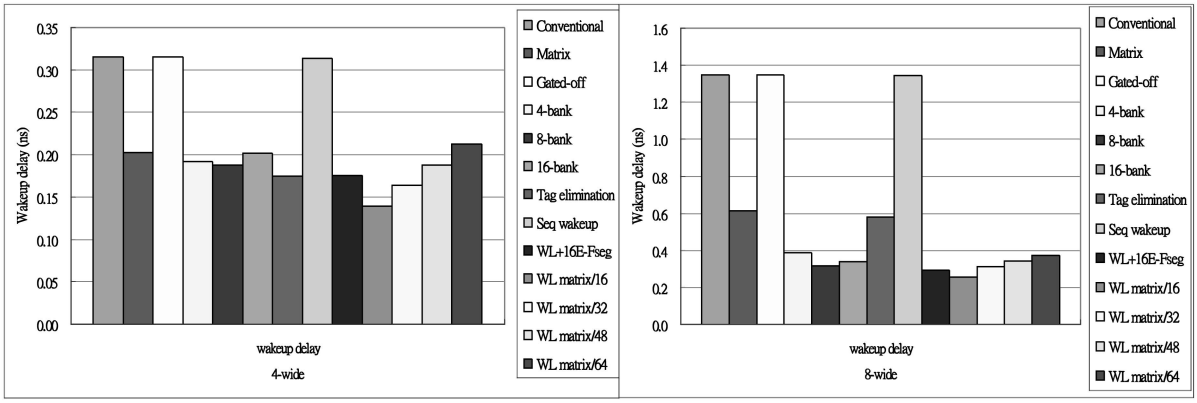

Fig. 14. Wake-up latencies of different wake-up approaches (ns).

inherent nature of the CAM structure, the power consumption of the gated-off design is still high, as shown in the third bars.

The second bars show that the bit matrix design takes about 25 percent (IW_{256}) and 29 percent (IW_{128}) of the power consumed by the CAM scheme. This significant improvement comes from the nature of the bit matrix. This design replaces the CAM structures, which are associative lookup units, with the RAM-like structures that are direct access units.

The fourth to sixth bars show the power consumption for the banked wake-up design [1]. Overall, the 4-bank design improves the power consumption by reducing the load capacitance of tag driving in the CAM structures. However, the overhead for the extra tag line and driver transistors becomes significant in the wider-banked design. This overhead of the extra tag line deteriorates 3-29 percent power consumption of the conventional design for the 16-bank design.

The power consumption of the tag elimination and sequential wake-up designs are shown in the seventh and eighth bars. The configuration of the tag-reduced design is equivalent to half of the entries of the CAM scheme; thus, this design saves 53 percent (IW_{256}) and 56 percent (IW_{128}) power consumption of the conventional design. In contrast, although the sequential wake-up design wakes up instructions in two phases, this design still drives two monolithic CAM structures, as the conventional design does. It is measured that the power consumption of the sequential wake-up design is similar to that of the conventional CAM design.

The power consumption of the proposed CAM-based wake-up locality scheme is shown in the next bars. It is measured to be only 24 percent (IW_{256}) and 35 percent (IW_{128}) that of the conventional CAM scheme. This excellent energy saving comes from the wake-up range limitation. Most needless tag driving and tag matching are filtered; thus, this design is highly efficient in terms of energy usage.

The last four bars show the power consumption of the matrix-based wake-up locality design with different configurations. Because the nature of the RAM-based design is energy efficient and only the necessary segments are activated during the wake-up process, the power consumption of this design is much smaller than that of other designs. This design achieves excellent energy saving that takes only 6-29 percent power consumption of the conventional CAM scheme for the 4-wide configuration and 2-11 percent for the 8-wide configuration.

4.4 Wake-Up Latency

The wake-up delay of the CAM-based wake-up designs can be summarized as follows:

$$T_{CAM} = T_{tagread} + T_{tagdrive} + T_{tagmatch} + T_{matchOR},$$

where $\mathrm{T_{tagread}}$ is the time for reading the destination tag from the tag RAM, $\mathrm{T_{tagdrive}}$ is the time for driving the tag into the CAM structure, $\mathrm{T_{tagmatch}}$ is the time spent by the match circuit in pulling the match line low, and $\mathrm{T_{matchOR}}$ is the time for performing a logical OR operation with the match lines.

The wake-up latency of the matrix-based scheme can be represented as follows:

$$T_{bmRAM} = T_{wordline} + T_{bitline} + T_{senseamp},$$

where $\mathrm{T_{wordline}}$ is the delay as the word line driver drives the grant signal into the bit-map RAM, $\mathrm{T_{bitline}}$ is the time for activating the bitline, and $\mathrm{T_{senseamp}}$ is the time for amplifying the bitline.

Fig. 14 shows the wake-up latencies of the wake-up designs in the 4-wide and 8-wide processors. The wake-up latencies of the conventional CAM design and gated-off design are presented in the first and the third bars. Since the gated-off scheme only gates the match lines in the empty and ready entries from activities and this does not affect the critical path of the wake-up operation, the wake-up latency of the gated-off design is the same as that in the conventional CAM design.

Compared to these CAM-based schemes, the bit matrix design is efficient in terms of wake-up latency because the nature of the RAM-like structure is much simpler than that of the CAM structure. The second bar shows that the wake-up latency of the bit matrix design is 64 percent and 46 percent that of the CAM scheme for the 4-wide and 8-wide processors.

Compared to the conventional CAM design, the banked design improves 36-40 percent wake-up latency in the 4-wide processor and improves 71-76 percent wake-up

TABLE 4
The Number of SRAM Bits Used

	128-entrry IW	Ratio	256-entry IW	Ratio
Conventional CAM-based	128 × 7 × 2 × 2 + 256 (Rdy) = 3840	1	256 × 8 × 2 × 2 + 512 (Rdy) = 8704	1
Conventional matrix-based	128 ×128 × 2 = 32768	8.53	256×256×2 = 131072	15.06
WL16 (SS= 16)	128 × 5 × 2 × 2 + 16 × 7 × 2 × 2 + (256 + 32) = 3296	0.86	256 × 5 × 2 ×2 + 16× 8× 2 ×2 + (512 + 32) = 6176	0.71
WL matrix/16	128 × 32 × 2 = 8192	2.13	256 × 32 × 2 = 16384	1.88
WL matrix/64	128 × 80 × 2 = 20480	5.33	256 × 80 × 2 = 40960	4.71

latency in the 8-wide processor. This improvement comes from the smaller banked segments that perform wake-up operations in parallel.

The tag elimination design has an equivalent half of the CAM structure to that of the conventional design for the wake-up operation. The wake-up latency of this design is measured to be 45 percent and 57 percent that of the conventional design for the 4-wide and 8-wide processors. In contrast to the tag elimination design, the sequential wake-up design has almost the same wake-up latency as that of the conventional design because the critical path of the sequential wake-up design is the same as that of the conventional design.

The proposed CAM-based wake-up locality design performs much faster than the conventional design. The proposed design is measured to be 44 percent (IW_{128}) and 78 percent (IW_{256}) faster than the conventional design and 49 percent faster than the tag elimination design for the 8-wide processor. Compared to the banked design, our proposed design is measured to be 7-8 percent faster in the 4-wide and 8-wide processors, respectively. This advantage comes from the fact that only the necessary segments are activated during the wake-up process. In addition, the matrix-based wake-up locality design achieves the best wake-up delay compared to the other designs when the wake-up range is limited to $\boldsymbol{i+16}$ instructions with $0 \leq \boldsymbol{i} \leq 15$.

5 Area Usage and Variant Designs

From the experimental results, we can observe that the major limiting factor for the scalability of the conventional CAM design is the complexity that induces significant overhead in terms of latency and power consumption. As for the bit matrix scheme, the limiting factor is its large area requirement. Table 4 shows the estimated memory requirements in terms of SRAM cells for the different wake-up designs. We assume that a typical CAM cell consists of two SRAM cells [45]. The proposed CAM-based wake-up locality design, the WL16 model, which has a wake-up coverage of $\boldsymbol{i+16}$ instructions for $0 \leq \boldsymbol{i} \leq 15$, uses fewer SRAM cells than the conventional designs despite the use of an Fseg. In contrast to the conventional CAM design, the CAM-based wake-up locality design is scalable with the issue window because only two reduced segments and one Fseg are searched during the wake-up process regardless of the window size. The proposed CAM-based WL design shows the advantages in the wake-up speed, power efficiency, and area cost with no performance compromise.

Both the WL_matrix/16 and WL_matrxi/64 configurations have shown good scalability with respect to the issue window size. Compared to the conventional matrix design, the matrix-based wake-up locality design has reduced the area cost drastically in addition to the advantages in wake-up speed and power dissipation. The matrix-based wake-up locality design with a wake-up range of $\boldsymbol{i+16}$ instructions $(0 \leq \boldsymbol{i} \leq 15)$ takes only 12.5 percent (IW_{256}) and 25 percent (IW_{128}) the area cost of the conventional matrix design.

Wake-up locality is most useful for large instruction windows because the design is based on the program's wake-up distances that are mostly less than 32. For each wake-up, there are two reduced segments and one Fseg that are searched. With 16-entry segments, this means that the number of entries which are searched is always limited to 48, no matter how large the issue window size is and, hence, the essence of the proposed WL-based scheme is the excellent scalability for large issue windows. Generally, this advantage gradually diminishes as the issue window size gets smaller. The WL-based scheme inherently does not provide useful improvement for a small window size, such as 32 entries or around 48 entries, despite the split structure. For example, in a 32-entry issue window, two reduced segments can be used without the Fseg. However, for each wake-up, the two reduced segments are searched; this is fundamentally the same as in a 32-entry single monolithic structure. Nevertheless, with a large issue window, the wake-up locality design can always bound the number of wake-up operations to a split system of three 16-entry segments without IPC loss.

5.1 Variant of CAM-Based Wake-Up Locality Design

In this section, we present the variants of the CAM-based wake-up locality designs by changing the segment size and wake-up coverage. One change is to use a smaller segment size such as eight entries while maintaining the same wake-up coverage as the 16-entry segment example shown previously. The wake-up coverage of this variant can be described as $\boldsymbol{i+24}$ instructions with $0 \leq \boldsymbol{i} \leq 7$, denoted as

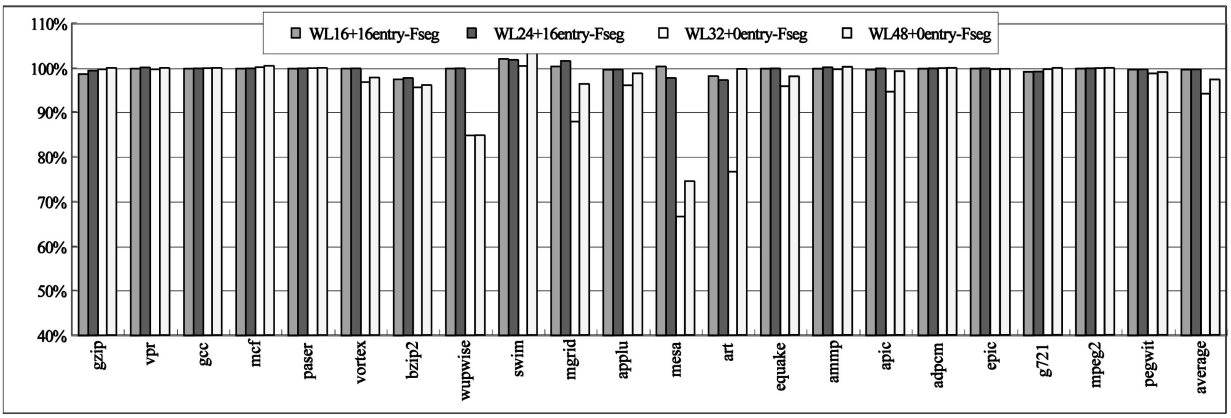

Fig. 15. The normalized performance of the 8-wide 256-entry processor for different wake-up locality designs.

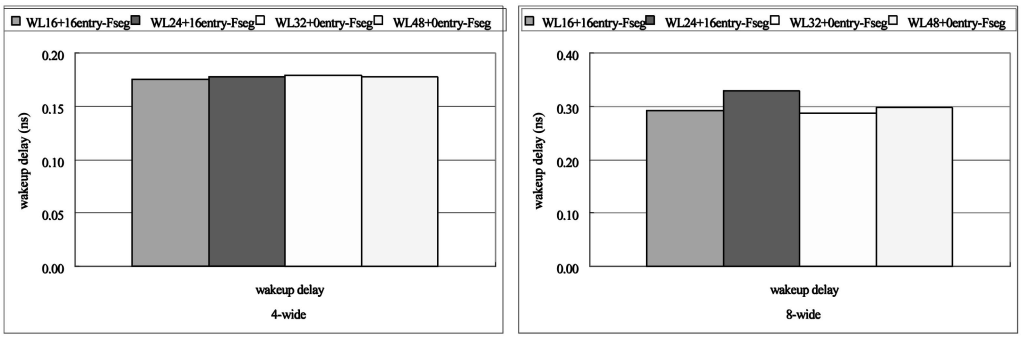

Fig. 16. Wake-up latency for the different wake-up locality designs.

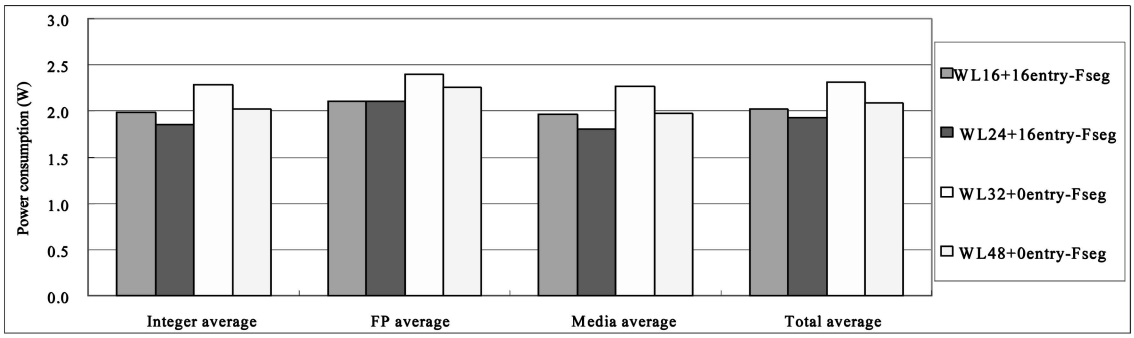

Fig. 17. Power consumption of the wake-up locality designs for an 8-issue 256-entry processor (w).

WL24. Other models include the wake-up coverage of $i + \mathbf{32}$ instructions $(0 \leq i \leq 31)$, WL32, and $i + \mathbf{48}$ instructions $(0 \leq i \leq 15)$, WL48. These two models have the same maximum wake-up coverage but use different segment sizes. Fig. 15 shows the IPC comparisons of these variants with the previous example, $i + \mathbf{16}$ instructions with $0 \leq i \leq 15$ (WL16). The first two bars show that the WL16 and WL24 models, each with a 16-entry Fseg, have achieved about the same IPC without degradation. For the WL32 and WL48 models, no Fseg is used since their maximum wake-up coverage is up to 63 instructions. On the average, the WL32 model has lost about 5 percent of the IPC, whereas the WL48 model has lost around 2 percent. These two models experience serious IPC loss for some of the benchmark programs, for example, wupwise and mesa, because of the lack of the Fseg.

Figs. 16 and 17 show comparisons of wake-up latency and power consumptions, respectively. As noted, there is no obvious difference among these variant designs. Comparing the IPC, power usage, and wake-up latency, it is found that the WL16 and WL24 models are the preferred configurations. However, the WL16 model requires less circuitry for the distance code comparisons, making the WL16 model most attractive to use.

5.2 Other Multiple-Bank CAM-Based Wake-Up Logic

Another way of doing multiple-bank wake-up design is to assign each segment a number as its wake-up address, which simply comes from the high-order bits of the source tag. To access the segments, the bits of the tag, source tag or result tag, are divided into two parts: index (wake-up address) and reduced tag. The index is the high-order bits of the tag that are used to select a segment. The other part, reduced tag, is the

low-order bits of the tag that are used as input for the wake-up logic for associative lookup. During the wake-up process, the result tag is compared with only the source tags that have the same wake-up address, that is, the same high-order bits. In this way, the instructions that have the same wake-up address are dispatched to the same segment.

During the wake-up process, the wake-up operation is performed only in the selected segment. The high-order bits (wake-up address) of the result tag are used to select one of the segments. The low-order bits of the result tag are used as the input for the wake-up logic to match with the source tags in this segment. The source tags in the other segments are not involved in this wake-up operation. This approach achieves the same effect as the wake-up locality design in the respect that only the selected segment is activated. A variant of the design can use the low-order bits of the tag to select a segment. However, segmenting the wake-up logic either with the high-order or the low-order bits of the source tag has several side-effects that may undermine the achievable IPC because of the stalling of instruction dispatch. This situation can be explained as follows.

When inserting instructions into the issue window, two scenarios will lead to the instruction not being dispatched to the selected segment. First, the same instruction has two wake-up addresses indexing to two different segments. Second, the segment selected by the wake-up address of the instruction is full; no more available entry of this segment can be allocated for the incoming instruction. Although this bank overflow occurs in the segments, others may suffer from bank underflow. Most importantly, the instruction order is lost and, thus, it is difficult to implement an age-based issue policy. Readers who are interested in more details of this design can refer to the work in [46]. Due to the loss of age-based issue order, the performance degradation in IPC is about 2.5-5 percent for the 4-wide and 8-wide processors. Although the wake-up latency is similar to those of the WL-based designs, the wake-up power consumption is about 1 to 2 watts higher than the WL-based designs due to the use of more supporting segments to reduce issue stalling.

6 Conclusion

In this paper, we show that most of the distances between two data dependent instructions are short in nature. It is found that 99 percent of the wake-up range is smaller than 31 instructions and 95 percent smaller than 16 instructions. Taking advantage of this wake-up spatial locality, two effective wake-up designs are proposed to improve the wake-up delay, power requirement, and scalability of the dynamic scheduler. For the CAM-based wake-up logic, the issue window is divided into IW/16 reduced segments, where IW is the issue window size. The wake-up range of a reduced segment is limited to $i + 16$ instructions, where $0 \leq i \leq 15$ to cover 95-99 percent of the wake-up operations. An Fseg of the same size is used to handle the wake-up operations for the instructions that are out of the limited wake-up range. This design significantly improves 44-78 percent of the wake-up delay and saves 65-76 percent power consumption compared to the conventional CAM scheme. Next, the matrix-based wake-up locality design, which limits the wake-up range to $i + 16$ instructions, requires only 12.5-25 percent area usage of the monolithic matrix design. Applying wake-up locality to the CAM-based and matrix-based wake-up logic has shown excellent scalability because the number of the activated segments during the wake-up process remains the same, regardless of the issue window size. In conclusion, the proposed wake-up designs remove the limiting factor from the dynamic scheduler and enable the processor to employ a more sophisticated scheduler for performance.

Acknowledgments

The authors thank all the reviewers for their helpful suggestions that strengthen the paper. Chia-Jung Hsu contributed in part of the simulations. This work was supported in part by the National Science Council, Taiwan, Grant NSC 94-2220-E-006-008.

References

[1] S. Palacharla, N.P. Jouppi, and J.E. Smith, "Quantifying the Complexity of Superscalar Processors," Technical Report CS-1328, Univ. of Wisconsin-Madison, May 1997.
[2] K. Wilcox and S. Manne, "Alpha Processors: A History of Power Issues and a Look to the Future," *Proc. 32nd Ann. Int'l Symp. Microarchitecture, Cool Chips Tutorial,* Nov. 1999.
[3] A. Kumar, "The HP PA8000 RISC CPU," *IEEE Micro,* vol. 17, no. 2, pp. 27-32, Apr. 1997.
[4] G. Hinton et al., "The Microarchitecture of the Pentium 4 Processor," *Intel Technology J.,* Feb. 2001.
[5] K.C. Yeager, "MIPS R10000 Superscalar Microprocessor," *IEEE Micro,* vol. 16, no. 2, pp. 28-40, Apr. 1996.
[6] R.E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro,* vol. 19, no. 2, pp. 24-36, Mar./Apr. 1999.
[7] M. Butler and Y.N. Patt, "An Investigation of the Performance of Various Dynamic Scheduling Techniques," *Proc. 25th Ann. Int'l Symp. Microarchitecture (MICRO '92),* pp. 1-9, Dec. 1992.
[8] S.T. Srinivasan and A.R. Lebeck, "Load Latency Tolerance in Dynamically Scheduled Processors," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO '98),* pp. 148-159, Dec. 1998.
[9] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report,* vol. 9, no. 2, pp. 1-7, Feb. 1995.
[10] S.P. Song, M. Denman, and J. Chang, "The PowerPC 604 RISC Microprocessor," *IEEE Micro,* vol. 14, no. 5, pp. 8-17, Oct. 1994.
[11] L. Gwennap, "HAL Reveals Multichip SPARC Processor," *Microprocessor Report,* vol. 9, no. 3, pp. 1-7, Mar. 1995.
[12] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach,* second ed. Morgan Kaufmann, 1996.
[13] M. Brown, J. Stark, and Y. Patt, "Select-Free Instruction Scheduling Logic," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO '01),* pp. 204-213, Dec. 2001.
[14] M. Goshima et al., "A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO '01),* pp. 225-236, Dec. 2001.
[15] R. Ho, K.W. Mai, and M.A. Horowitz, "The Future of Wires," *Proc. IEEE,* vol. 89, pp. 490-504, Apr. 2001.
[16] M.S. Hrishikesh, N.P. Jouppi, and K.I. Farkas, "The Optimal Useful Logic Depth per Pipeline Stages Is 6-8 FO4," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA '02),* pp. 14-24, May 2002.
[17] D. Folegnani and A. González, "Energy-Effective Issue Logic," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA '01),* pp. 230-239, July 2001.
[18] M.A. Ramírez et al., "A Simple Low-Energy Instruction Wakeup Mechanism," *Proc. Int'l Symp. High-Performance Computing (ISHPC '03),* pp. 99-112, Oct. 2003.
[19] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing Power Requirements of Instruction Scheduling through Dynamic Allocation of Multiple Datapath Resources," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO '01),* pp. 90-101, Dec. 2001.

[20] J. Abella and A. González, "Power-Aware Adaptive Issue Queue and Register File," *Proc. Int'l Conf. High-Performance Computing (HiPC '03)*, Dec. 2003.
[21] D.H. Albonesi, "Dynamic IPC/Clock Rate Optimization," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '98)*, pp. 282-292, June 1998.
[22] A. Buyuktosunoglu et al., "A Circuit Level Implementation of an Adaptive Issue Queue for Poweraware Microprocessors," *Proc. Great Lakes Symp. VLSI (GLSVLSI '01)*, pp. 73-83, Mar. 2001.
[23] S. Dropsho et al., "Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power," *Proc. 11th Parallel Architectures and Compilation Techniques*, pp. 141-152, Sept. 2002.
[24] D. Ernst and T.M. Austin, "Efficient Dynamic Scheduling through Tag Elimination," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA '02)*, pp. 37-46, May 2002.
[25] J.J. Sharkey et al., "Instruction Packing: Reducing Power and Delay of the Dynamic Scheduling Logic," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '05)*, pp. 30-35, Aug. 2005.
[26] I. Kim and M.H. Lipasti, "Half-Price Architecture," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '03)*, pp. 28-38, June 2003.
[27] A. Aggarwal et al., "Defining Wakeup Width for Efficient Dynamic Scheduling," *Proc. Int'l Conf. Computer Design (ICCD '04)*, pp. 36-41, Oct. 2004.
[28] D. Ernst, A. Hamel, and T. Austin, "Cyclone: A Broadcast-Free Dynamic Instruction Scheduler with Selective Replay," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '03)*, pp. 253-262, June 2003.
[29] J. Hu, N. Vijaykrishnan, and M. Irwin, "Exploring Wakeup-Free Instruction Scheduling," *Proc. Int'l Symp. High Performance Computer Architecture (HPCA '04)*, pp. 232-241, Feb. 2004.
[30] A.R. Lebeck et al., "A Large, Fast Instruction Window for Tolerating Cache Misses," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '02)*, pp. 59-70, May 2002.
[31] B. Fields, S. Rubin, and R. Bodík, "Focusing Processor Policies via Critical-Path Prediction," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA '01)*, pp. 74-85, July 2001.
[32] E. Brekelbaum et al., "Hierarchical Scheduling Windows," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO '02)*, pp. 27-36, Nov. 2002.
[33] D.S. Henry, B.C. Kuszmaul, G.H. Loh, and R. Sami, "Circuits for Wide-Window Superscalar Processors," *Proc. Ann. Int'l Symp. Computer Architecture (ISCA '00)*, pp. 236-247, June 2000.
[34] K.S. Hsiao and C.H. Chen, "An Efficient Wakeup Design for Energy Reduction in High-Performance Superscalar Processors," *Proc. Int'l Conf. Computing Frontiers (CF '05)*, pp. 353-360, May 2005.
[35] D.V. Ponomarev et al., "Energy-Efficient Issue Queue Design," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 11, pp. 789-800, Oct. 2003.
[36] M. Huang, J. Renau, and J. Torrellas, "Energy-Efficient Hybrid Wakeup Logic," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '02)*, pp. 196-201, Aug. 2002.
[37] R. Canal and A. González, "A Low-Complexity Issue Logic," *Proc. Int'l Conf. Supercomputing (ICS '00)*, pp. 327-335, May 2000.
[38] R. Canal and A. Gonzalez, "Reducing the Complexity of the Issue Logic," *Proc. Int'l Conf. Supercomputing (ICS '01)*, pp. 312-320, June 2001.
[39] S. Palacharla, N.P. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '97)*, pp. 206-218, June 1997.
[40] P. Michaud and A. Seznec, "Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors," *Proc. Int'l Symp. High Performance Computer Architecture (HPCA '04)*, pp. 27-36, Jan. 2001.
[41] S.E. Raasch, N.L. Binkert, and S.K. Reinhardt, "A Scalable Instruction Queue Design Using Dependence Chains," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '02)*, pp. 318-329, May 2002.
[42] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. Ann. Int'l Symp Computer Architecture (ISCA '00)*, pp. 83-94, June 2000.
[43] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, v2.0," Technical Report CS-1342, Univ. of Wisconsin-Madison, June 1997.
[44] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: A Tool for Evaluating Multimedia and Comm. Systems," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO '97)*, pp. 330-335, Dec. 1997.
[45] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Architecture: A Tutorial and Survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712-727, Mar. 2006.
[46] K.S. Hsiao and C.H. Chen, "Improving Scalability and Complexity of Dynamic Scheduler through Wakeup-Based Scheduling," *Proc. Int'l Conf. Computer Design*, Oct. 2006.

Chung-Ho Chen received the MS degree in electrical engineering from the University of Missouri, Rolla, in 1989 and the PhD degree in electrical engineering from the University of Washington, Seattle, in 1993. In 1993, he became a faculty member in the Department of Electronic Engineering at National Yunlin University of Science and Technology. In 1999, he joined the Department of Electrical Engineering, National Cheng Kung University, where he is currently a professor. His research interests include advanced computer architecture, video technology, and network storages. He is a coholder of a US patent on a multicomputer cluster-based processing system and of a Republic of China patent on a multiple-protocol storage structure. He was the technical program chair of the 2002 VLSI Design/CAD Symposium held in Taiwan. He is a member of the IEEE.

Kuo-Su Hsiao received the BS degree in computer and communication engineering from the National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan, in 1999 and the MS degree in electrical engineering and the PhD degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 2001 and July 2006, respectively. His research interests include computer architecture, low-power processor, and VLSI design.