Scheduler Optimization by Exploring Wakeup Locality

Kuo-Su Hsiao and Chung-Ho Chen
Department of Electrical Engineering, National Cheng Kung University
No.1, Ta-Hsueh Road, Tainan 701, Taiwan
newjimmy@ee.ncku.edu.tw, chchen@mail.ncku.edu.tw

***Abstract*- In a high-performance superscalar processor, the instruction scheduler often comes with poor scalability and high complexity due to the expensive instruction wakeup operation. Using detailed simulation-based analyses, we find that the wakeup distances between two dependent instructions are short. By exploiting this wakeup locality, an effective wakeup design is proposed to improve the speed, power, and scalability of the dynamic scheduler. By limiting the wakeup range of instructions, load capacitance and match activities on the scheduler's critical path can be reduced. The architectural level simulation and circuit-level timing analyses show that the proposed design saves 65-76% of the power consumption, reduces 44-78% in the wakeup latency with negligible (less than 1%) performance degradation. The results also show that the proposed design is excellent in scalability.**

I. INTRODUCTION

The complexity of dynamic instruction schedulers is one of the most important issues in high-performance microprocessor design. To extract more instruction level parallelism (ILP) and boost instructions per cycle (IPC), superscalar processors tend towards the specification of using a large issue window and wide issue width. Consequently, the dynamic scheduler required for out-of-order execution becomes more complex and less scalable. The complex scheduler consumes a lot of energy and may slow down the clock cycle time.

In particular, the complexity of the scheduler comes mainly from the wakeup logic that traces the instruction dependences and wakes the instructions up when their source operands become available. The wakeup logic is typically implemented by using the content-addressable memories (CAM) 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 wakeup speed due to considerable circuit activities and heavy load capacitance.

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 wakeup logic. Although a pipelined dynamic scheduler can increase the clock frequency, the operations of instruction wakeup and instruction selection should be an atomic operation to avoid significant performance degradation. Recent study has shown that the latencies associated with the wakeup and selection form the critical path of the pipeline stages [1]. The wakeup latency increases significantly with both the issue width and the window size; and the wakeup logic dominates the latency for the scheduler [1]. Increasing the window size and issue width will continue to increase the burden to the clock cycle time.

For the energy consideration, the power consumption associated with the CAM-based 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; it is responsible for 46% of the total processor power [2]. Similarly, the out-of-order scheduler of the Intel Pentium 4 processor accounts for 40% of the total power consumption. As a result, the CAM-based wakeup logic not only slows down the clock speed but also shifts more power budget to the scheduler.

Analyses reveal that 96% of the wakeup distances between the dependent instructions are in a range of 16 instructions. Based on this observation, the proposed optimization limits the wakeup range for the instructions in the issue window to improve the complexity and the scalability of the scheduler. In particular, the proposed design divides the monolithic wakeup logic into multiple segments and the wakeup ranges for the segments are limited to a smaller range. In this way, only the segments that are selected by the result tags are activated during the wakeup process. Since most of the wakeup distances are short, this technique effectively removes the needless wakeup operations to reduce the power consumption and wakeup delay.

The remainder of this paper is organized as follows. Section II explains the limitations of the current wakeup logic used in dynamic schedulers and provides a brief review of related works. Section III details the proposed wakeup designs. Section IV presents the experimental methodology and the evaluation results. Finally, Section V concludes this paper.

II. BACKGROUND

This section gives the background of the wakeup logics, including the current design and related works.

A. Limitations of the current wakeup logic

Figure 1 shows the conventional implementation of wakeup logic based on the CAM structure [1]. This wakeup logic employs two CAM structures to match the result tags with the left and right source tags of the instructions in the issue window. Each entry of the CAM stores the source tag of the instruction. Besides, the ready bits (Rdy L and Rdy R) are employed to indicate whether their corresponding operands are available or not. For the wakeup 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 a 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 and latency. During the wakeup process, many tag lines are driven and the load capacitances on the tag lines are heavy because the match circuits of all the entries in the CAM must be driven. Additionally, many match lines are

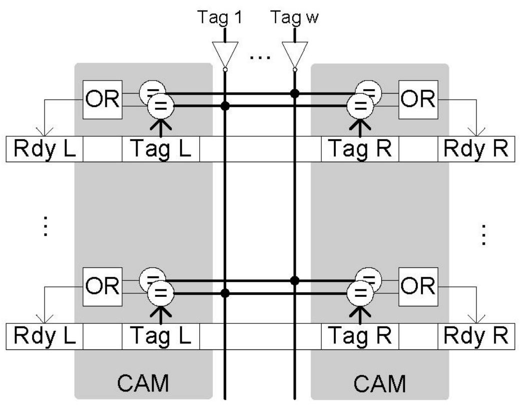

Figure 1: The conventional wakeup logic.

activated in wakeup operation no matter it is a match or not. Both the tag driving and the tag matching consume a lot of energy and slow down the wakeup speed.

In an effort to improve the instructions per cycle (IPC), scheduler designs often employ larger window and aggressive issue width. In other words, larger window leads to larger wakeup logic and larger wakeup logic leads to heavier load capacitances and more match activities. At the same time, wider issue width leads to driving more tag buses into the wakeup logic. We can see that increasing the window size and issue width leads to larger power consumption and slower scheduling speed. As a result, the scheduler can not scale well with the increasing of window size and issue width.

B. *Related works*

There have been many efforts to reduce the complexity of dynamic schedulers, many of which can be used in combination with the proposed technique.

Some current designs segment the monolithic issue window into multiple banks to improve wakeup delay, while the result tags need to be broadcast to all the banks [1][4]. This design induces extra wakeup delay and power consumption due to the additional driver-transistors and tag buses.

Hrishikesh et al. proposed a pipelined-wakeup design that segments the issue window and wakes up the instructions in the segments in multiple sequential cycles [3]. However, all the segments stills need to be searched; besides the dependent instructions can be issued back to back only if they are in the first segment.

Some approaches dynamically manage the sizes of the issue window and turn off the useless entries [4-9]. These designs improve the power consumption of the scheduler with extra dynamic managers that may complicate the scheduler on the other hand. In [4], Folegnani and González also presented a gate-off technique that disables the useless (empty and ready) entries of the issue window from tag matching

Ernst and Austin proposed a scheduler that employs less tag comparators to reduce the complexity of the scheduler. This scheduler also has a last tag speculator to reduce the frequency of tag matching [10]. Kim and Lipasti proposed a sequential wakeup mechanism to reduce the complexity of scheduler [11]. This mechanism places the last-arrival operand into the fast wakeup logic and wakes up left and right source operands of an instruction in two sequential steps. Besides, some works [12-14] employ two-level issue window to reduce the complexity of the scheduler. The critical instructions are dispatched to the small and fast issue window and the non-critical 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 wakeup designs employ the custom components rather than the CAM structures. Goshima et al. presented a wakeup design that uses bit matrix structures instead of the CAM structures [15]. Henry et al. presented a cyclic segmented prefix (CSP) circuit to improve the performance of wakeup logic [16]. Hsiao and Chen presented a wakeup design, which pre-decodes the source tags and matches the decoded outputs directly with the grant lines, to improve the wakeup speed and power consumption [17]. Ponomarev et al. used three techniques, efficient comparators, 0-B encoding, and bitline segmentation, to reduce the energy dissipation of the issue window [18].

Finally, some designs reduce issue logic complexity through index-based techniques, using pointers to connect the producer instructions and consumer instructions [19-21]. Several works reduce the complexity of scheduler by pre-scheduling dependent instructions into data-flow based issue window [22-25].

Some of these designs improve the scheduler in one or more issues (power consumption, speed, performance, etc.) but induce overhead on the others. And some optimizations improve the scheduler with extra large or complex units. In contrast, our designs improve the complexity and scalability of the scheduler with no extra large or complicated unit and the performance degradation of the proposed design is negligible.

III. Wakeup Design Optimization

In this section, we observe an interesting program behavior that most of the distances between two data dependent instructions are short. Based on this observation, a wakeup optimization is proposed to optimize the latency, energy, and scalability of the dynamic scheduler.

A. *Wakeup locality*

The wakeup locality is an inherent feature of two data dependent instructions in programs. In programs, the consumer instruction is often close to the producer instruction. This program characteristic when referring to the wakeup operation in an out-of-order execution processor is called the wakeup locality. The wakeup locality is measured by the instruction count between two data dependent instructions. The instruction count is also referred to as the wakeup distance.

To quantify the degree of the distance between two dependent instructions, a dynamic scheduled processor was simulated. When instructions were dispatched into the issue window, the instruction count between two dependent instructions was counted. Figure 2 shows the runtime distribution of the wakeup distance for all of the wakeup operations of the benchmark programs. The statistics are

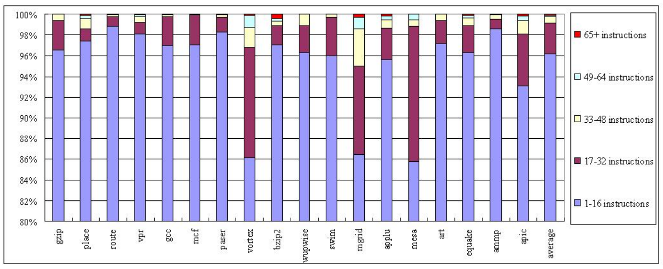

Figure 2: Runtime distribution of the wakeup distances for a 4-wide 128-entry processor.

based on a 4-wide processor with a 128-entry issue window (the simulation environment for the experiment is presented in the later section). Results are shown for seven of the integer programs and nine of the floating point programs of the SPEC2000 benchmark suite.

As observed, 96% of the wakeup distances are within the range of 16 instructions on average. And less than 1% of the wakeup operations come with the wakeup distances larger than 32 instructions. The wakeup locality is due to the fact that dependent instructions are often arranged in proximity to improve the data transfer and register utilization. For the integer benchmark programs, the wakeup distances between two dependent instructions are shorter than those of the floating point programs on average. This is because the integer programs tend to have fewer global value communications and smaller subroutines.

B. *Exploring wakeup locality in wakeup logic*

The observation on the wakeup locality motivates the optimization of the wakeup logic. In this section, we present an efficient wakeup design taking the advantage of wakeup locality.

It is possible to take the advantage of using wakeup locality, if only a few entries of the wakeup logic are searched during the wakeup process. Conventional wakeup design handles the wakeup operations with the wakeup distance up to the issue window size. Due to the heavy load capacitance and considerable circuit activities, this conventional design has poor scalability and is inefficient both in terms of energy and speed. Since most of the wakeup distances of wakeup operations are short, it is not necessary to search all the source tags in the wakeup logic during the wakeup process.

The proposed design limits the wakeup range for the wakeup operations. This design employs multiple small segments and the wakeup ranges of the segments are limited. Each segment handles the wakeup operations only for the result tags that are in its wakeup range. The basic idea of this design is to reduce the load capacitance of tag driving and the circuit activities of tag matching by matching the result tags with the source tags only in short wakeup distance.

Figure 3 shows the example of the proposed wakeup design that limits the wakeup range of the wakeup operations to 16-31 instructions. Different from the conventional design, multiple small CAM structures are employed for the range-limited wakeup operation. The segments in this design are classified into two types, full segment and reduced segment. The reduced segments support the wakeup operations only

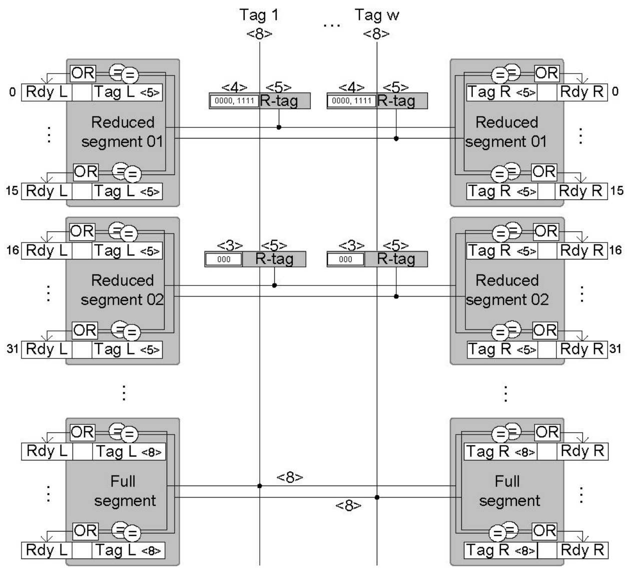

Figure 3: An example of the proposed wakeup design for a 256-entry issue window.

Table 1: Distance codes and corresponding wakeup ranges of the reduced segments for a 256-entry issue window.

Segment	S 0	S 1	S 2	S 3	…	S15
Distance codes	0000, 1111	000	0001, 0010	001	…	111
wakeup range	0-15, 240-255	0-31	16-47	32-63	…	224-255

for the instructions having the wakeup distances in the limited wakeup range. The full segment is used to handle the wakeup operations for the instructions that are out of the limited wakeup range.

The reduced segment each consists of 16 entries of the CAM. Each entry is assigned an entry number in the sequential order as that in the conventional design. To limit the wakeup range of the reduced segment, the inputs for this segment are limited to only the result tags that have the values in the limited wakeup range of this segment.

The limited wakeup ranges for the reduced segments are shown in Table 1. The wakeup range for each reduced segment is selected to be the following 32 instructions: the 16 instructions whose result tag numbers are the same as the entry numbers of the selected reduced segment, and the 16 preceding instructions whose result tag numbers are the 16 numbers before the first entry of this segment. For example, the wakeup range of the first reduced segment shown in Figure 3 is the numbers from 0 to 15, which are the same as the entry numbers of the segment, and the numbers 240-255, which are the 16 numbers before the first entry (entry 0) of this segment.

In this way, the wakeup range for the instruction in the first entry of the reduced segment is the 16 preceding instructions before this instruction, and by analogy, the wakeup range for the instruction in the last entry is the 31 preceding instructions before this instruction.

Since the wakeup ranges of the reduced segments are limited, the bit length of the inputs (result tags) and the bit length of the source tags can be reduced. In the example of Figure 3, the least significant 5 bits of the 8-bit result tags are used as inputs for the reduced segments and the source tag fields in the reduced segment store only the 5 low-order bits of the source tags.

In addition to the reduced segments, a full segment, shown in the bottom of Figure 3, is employed to handle the wakeup operations for the instructions with the wakeup distances out of the range supported by the reduced segments. The full segment is a small segment of the conventional wakeup logic that handles the wakeup operations without the constraint on wakeup range. Since only 4% of the wakeup distances of the dynamic instructions are out of the limited range, a 16-entry full segment can handle the wakeup operations for the out-of-range instructions well. More discussion about the trade-off between the performance and the entry quantities of the full segment will be presented in next section.

The access to the proposed wakeup design is a little different from the access to a conventional design. After rename, the instruction is allocated a destination tag (entry number) to index 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.

The source tag of the instruction with the wakeup distance in the wakeup range of the allocated segment is inserted into the allocated reduced segment. On the other hand, the source tag of the instruction with a wakeup distance out of the range of the allocated segment is inserted into the full segment.

It is easy to determine whether the wakeup distance is out of range or not by checking the most significant bits (distance code) of the source tag. For example, the most significant three or four bits of the source tag are matched with the distance code(s) shown in Table 1 according to the allocated segment. If it is a match, the wakeup distance of this source tag is in the wakeup range of the allocated reduced segment and then the least significant 5 bits of this source tag are inserted into the allocated entry. If the wakeup distance is out of the range, the source tag is inserted into the full segment.

During the wakeup process, the result tag is always used as input for the full segment; however, the result tag is only used as input for the reduced segments that have the same distance codes the same as the result tag.

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 wakeup process. These factors significantly improve the power consumption and wakeup latency of the scheduler. Another advantage of this design is the excellent scalability. No matter what the issue window size is, the number of the activated segments remains the same during the wakeup process.

IV. EXPERIMENTAL EVALUATION AND ANALYSIS

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

Table 2: 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.	

A. *Experimental methodology*

The power consumption and IPC results of the evaluated designs were obtained through architectural simulation, which was conducted by using Wattch [26] and SimpleScalar [27] toolsets. These execution-driven simulators simulate a superscalar processor with two-level caches, branch predictors, dynamic scheduler, and et al. by performing cycle by cycle instruction-level simulation, including execution down any speculative path until a branch misprediction is detected.

Table 2 lists the architectural parameters for the 4-wide and 8-wide superscalar processors. In Wattch, the CAM cell of the evaluated designs was based on the CAM model in [1]. The other configurations for the Wattch include 1GHz clock frequency, 1.8V voltage, and 0.18μm technology process.

The simulation results were collected from 7 integer and 9 floating point programs of the SPEC2000 benchmark suite. All the selected benchmark programs were compiled with full optimization (-O4). The test input set was used for the benchmark programs. The programs were fast-forwarded the first 0.5 billion instructions and the following 5 billion instructions were simulated.

To understand the effects on the wakeup delay, the circuit characteristics of the evaluated designs must be examined. The circuit models were extended from the one proposed by Ernst and Austin [10] and the timing results for the evaluated designs were extracted by using the Avant! Hspice tool. Finally, the CMOS transistors and wires were all conformed to the parameters of the TSMC 0.18μm process.

B. *Performance comparison*

Figure 4 presents the IPCs of the 4-wide and 8-wide processors that employ different wakeup logics. These results are normalized to the IPC of the baseline processor, which employs the conventional wakeup logic. The first bars show that the performances of the gated-off designs are the same as that of the conventional processor. The gated-off design only gates off the ready and empty entries of the CAM structures. This design changes no architectural configurations; thus the IPC result is the same as that of the conventional processor.

The second and third bars show the IPC drops due to the tag elimination design and sequential wakeup design. The tag

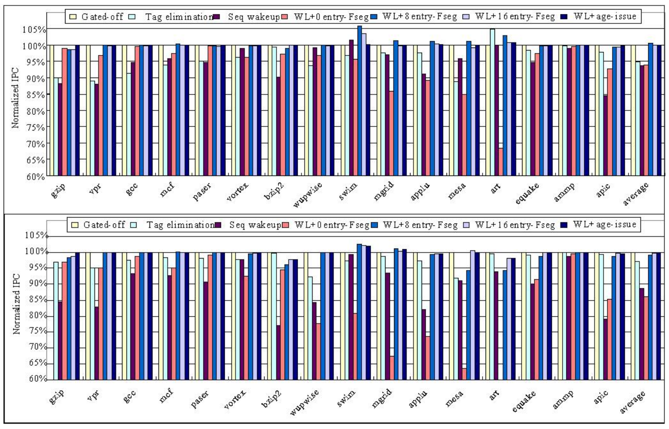

Figure 4: The normalized performance of the evaluated designs for the 4-wide (upper part) and 8-wide (lower part) processors.

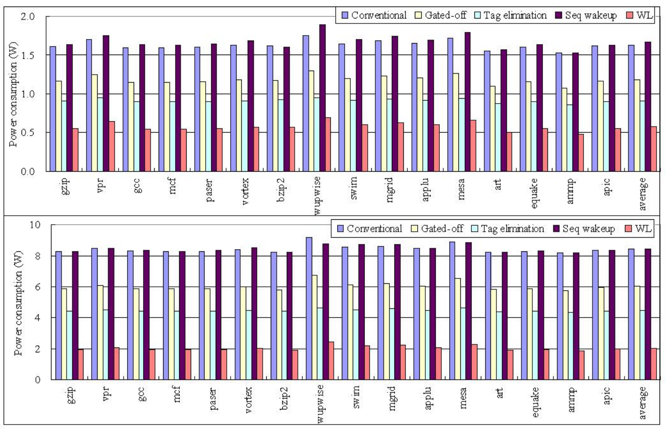

Figure 5: Power consumption of the evaluated designs for the 4-wide (upper part) and 8-wide (lower part) processors.

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 3-5% of IPC due to the issue policy and capacity conflicts. Since the quantity of entry is sufficient in the 8-wide configuration, the capacity conflicts occur less frequently. On the other hand, the IPC drop due to the sequential wakeup design is measured to be 6-11%. Obviously, waking instructions up in two sequential cycles induces non-negligible performance degradation.

There is almost no performance degradation due to the proposed wakeup locality (WL) design. The IPC loss is measured to be only 0.2-0.9% for the 8-wide processor and no IPC loss for the 4-wide processor as shown in the fifth and sixth bars. The slight IPC drop for the 4-wide processor comes from the extra dispatch stall when the full segment has no more available entry for the out-of-range instruction.

The fourth bars show the performance for the proposed design without the full segment. Although only 4% of the dynamic instructions are out of the wakeup range, the dispatch stalls due to these instructions induce 6-14% IPC drop. We can see that the 16-entry full segment is enough for avoiding this significant performance degradation for the 4-wide and 8-wide processors.

In addition, the last bars show the performance of the wakeup locality design with the oldest-first issue policy. The performance is the same as that of the WL design without aged-issue policy. In the wakeup locality design, since instructions are dispatched into the reduced and full segments, the instruction order can not be kept easily as that in the conventional design. During the instruction issue process, the proposed design can not do the oldest-first issue. The instructions in the reduced segments are assigned higher priority than those in the full segment. Although the instructions lost their program order, they are still in their age order in the reduced segments. Based on the position (entry number), the instructions in the reduced segments can be issued in the aged-issue policy. The performance of the WL design with the two-priority issue policy, shown in the fifth bars, is as good as that of the WL design with the oldest-first issue policy.

Interestingly, the wakeup locality design slightly out-performs than the baseline processor in some benchmark programs. This is because some instructions on the miss-predicted paths may be stalled in this design; thus these misprediction recoveries are avoided.

To summarize, the proposed design has a better performance than the tag elimination design and sequential wakeup design. Although the wakeup range is limited to 16 instructions, the WL design achieves almost no performance degradation due to the employment of the full segment. Besides, the employed two-priority issue policy can achieve the same performance of the aged-issue policy.

C. *Power consumption*

Figure 5 presents the power consumption for the wakeup logics in the 4-wide and 8-wide processors. The power consumption of the conventional design, shown at the left most bars, is found to be much higher than others. This is due to the heavy load capacitance and the surplus activities of the monolithic CAM structure. The gated-off design reduces 28-29% power consumption of the conventional design by gating the ready and empty entries from tag matching. Due to the inherent nature of the CAM structure, the power consumption of the gated-off design is still high as shown in the second bars.

The power consumption of the tag elimination and sequential wakeup designs are shown in the third and fourth bars. The configuration of the tag elimination design is equivalent to half of the entries of the conventional design; thus this design saves 44-47% power consumption of the conventional design. In contrast, although the sequential wakeup design wakes up instructions in two phases, this design still drives two monolithic CAM structures as the conventional design does. The sequential wakeup design improves no power consumption of the conventional design.

The power consumption of the proposed WL design is shown in the last bars. It is measured to be only 24-35% that of the conventional design. This excellent energy saving comes from the limitation of the wakeup range. Most needless tag driving and tag matching are avoided; thus this design is highly efficient in terms of energy usage.

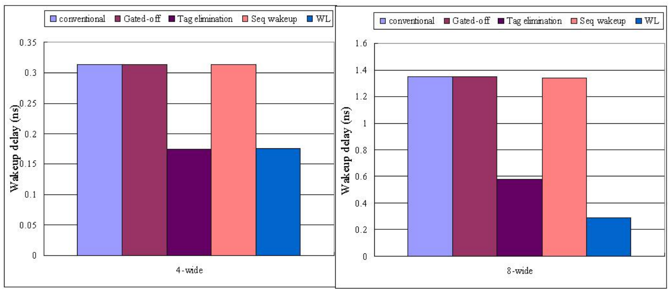

Figure 6: Wakeup latencies of different wakeup approaches (ns).

D. *Wakeup latency*

Figure 6 shows the wakeup latencies of the evaluated wakeup designs for the 4-wide and 8-wide processors. The wakeup latencies of the conventional design and gated-off design are presented in the first two bars. Since the gated-off scheme only gates the match lines in the empty and ready entries from activities that does not affect the critical path of the wakeup operation, the wakeup latency of the gated-off design is the same as that in the conventional design.

The tag elimination design has an equivalent half of the CAM structure to that of the conventional design for the wakeup operation. The wakeup latency of this design is measured to be 43-56 % that of the conventional design. As for the sequential wakeup design, this design only avoids driving the driver transistors of the second CAM structure in the first wakeup cycle. The improvement of wakeup latency is measured only 0.5% for the 4-wide processor.

The proposed wakeup locality design performs much faster than other designs. This advantage comes from that only necessary segments are activated during the wakeup process. The proposed designs is measured to be 44-78% faster than the conventional design and 50% faster than the tag elimination design for the 8-wide processor. We can see that the proposed design suits for a sophisticated scheduler in terms of the wakeup speed.

V. Conclusion

In this paper, we present the concept of wakeup locality in which most of the wakeup distances between two data dependent instructions are found to be short. Based on this, an effective wakeup design is proposed to improve the wakeup delay, power requirement, and scalability of the dynamic scheduler. The proposed wakeup locality design limits the wakeup range of the wakeup operations and activates only the necessary segments during the wakeup process. This design significantly improves 44-78% of the wakeup delay and saves 65-76% power consumption compared to the conventional wakeup logic without the performance degradation. Besides, the proposed design is excellent in scalability because the number of the activated segments during the wakeup process remains the same regardless of the issue window size. In conclusion, the proposed wakeup design removes the limiting factor from the dynamic scheduler and enables the processor to employ a more sophisticated scheduler for improving performance.

Acknowledgment

The work in this paper is in part supported by the National Science Council, Taiwan R.O.C., under NSC 94-2220-E-006-004.

References

[1] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Quantifying the Complexity of Superscalar Processors," University of Wisconsin-Madison, Tech. Rep. CS-1328, May 1997.
[2] K. Wilcox and S. Manne. "Alpha processors: A history of power issues and a look to the future," Cool Chips Tutorial, 32nd Annu. Int. Symp. Microarchitecture, Nov. 1999.
[3] M. S. Hrishikesh, N. P. Jouppi, and K. I. Farkas, "The optimal useful logic depth per pipeline stages is 6-8 FO4," in Proc. ISCA, May 2002, pp. 14-24.
[4] D. Folegnani and A. Gonzalez, "Energy-Effective Issue Logic", in Proc. ISCA, Jul. 2001, pp. 230-239.
[5] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources," in Proc. MICRO, Dec. 2001, pp. 90-101.
[6] J. Abella and A. González, "Power-Aware Adaptive Issue Queue and Register File," in Proc. Int. Conf. High-Performance Computing (HiPC), Dec. 2003.
[7] David H. Albonesi. "Dynamic IPC/Clock Rate Optimization," in Proc. ISCA, June 1998, pp. 282–292.
[8] A. Buyuktosunoglu et al., "A Circuit Level Implementation of an Adaptive Issue Queue for Poweraware microprocessors," in Proc GLVSLSI, Mar. 2001, pp. 73-83.
[9] S. Dropsho et al., "Integrating Adaptive On- Chip Storage Structures for Reduced Dynamic Power," in Proc. Parallel Architectures and Compilation Techniques, Sep. 2002, pp. 141-152.
[10] D. Ernst and T. M. Austin, "Efficient dynamic scheduling through tag elimination," in Proc. ISCA, May 2002, pp. 37-46.
[11] I. Kim and M. H. Lipasti, "Half-Price Architecture," in Proc. ISCA, Jun. 2003, pp. 28-38.
[12] A.R. Lebeck et al., "A Large, Fast Instruction Window for Tolerating Cache Misses," in Proc. ISCA, May 2002, pp. 59-70.
[13] B. Fields, S. Rubin, and R. Bodík, "Focusing Processor Policies via Critical-Path Prediction," in Proc. ISCA, Jul. 2001, pp. 74-85.
[14] E. Brekelbaum et al., "Hierarchical Scheduling Windows," in Proc. MICRO, Nov. 2002, pp. 27-36.
[15] M. Goshima et al., "A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors," in Proc. MICRO, Dec. 2001, pp. 225-236.
[16] D. S. Henry et al., "Circuits for Wide-Window Superscalar Processors," in Proc. ISCA, Jun. 2000, pp. 236-247.
[17] K.-S. Hsiao and C.-H. Chen, "An Efficient Wakeup Design for Energy Reduction in High-Performance Superscalar Processors," in Int. Con. Computing Frontiers (CF), May 2005. pp. 353-360.
[18] D. V. Ponomarev et al., "Energy-Efficient Issue Queue Design," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 11, pp. 789-800, Oct. 2003.
[19] M. Huang, J. Renau, and J. Torrellas, "Energy-Efficient Hybrid Wakeup Logic," in Proc. ISLPED, Aug. 2002, pp. 196-201.
[20] R. Canal and A. González, "A Low-Complexity Issue Logic," in Proc. ICS, May 2000, pp. 327-335
[21] R. Canal and A. Gonzalez, "Reducing the Complexity of the Issue Logic," in Proc. ICS, Jun. 2001, pp. 312-320.
[22] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in Proc. ISCA, Jun. 1997, pp. 206-218.
[23] P. Michaud and A. Seznec, "Data-flow prescheduling for large instruction windows in out-of-order processors," in Proc. HPCA, Jan. 2001, pp. 27-36.
[24] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, "A Scalable Instruction Queue Design Using Dependence Chains," in Proc. ISCA, May 2002, pp. 318-329.
[25] D. Ernst, A. Hamel, and T. Austin, "Cyclone: A Broadcast-Free Dynamic Instruction Scheduler with Selective Replay," in Proc. ISCA, Jun. 2003, pp. 253-262.
[26] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in Proc. ISCA, Jun. 2000, pp. 83-94.
[27] D. Burger and T. M. Austin,"The SimpleScalar tool set, version 2.0," University of Wisconsin-Madison, Tech. Rep. CS-1342, Jun. 1997.