Power-efficient and Scalable Load/Store Queue Design via Address Compression

Yi-Ying Tsai
Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan
magi@casmail.ee.ncku.edu.tw

Chia-Jung Hsu
Institute of Computer and Communication Engineering, National Cheng Kung University Tainan, Taiwan
jovi@casmail.ee.ncku.edu.tw

Chung-Ho Chen
Institute of Computer and Communication Engineering, National Cheng Kung University Tainan, Taiwan
chchen@mail.ncku.edu.tw

ABSTRACT
This paper proposes an address compression technique for load/store queue (LSQ) to improve the scalability and power efficiency. A load/store queue (LSQ) typically needs a fully-associative CAM structure to search the address for collision and consequently poses scalability challenges of power consumption and area cost. Using the proposed approach, the LSQ can reduce the area cost ranging from 32% to 66% and power consumption ranging from 38% to 71%, depending on the compression parameter. The approach can provide 3.08% overall processor energy reduction and causes only 0.22% performance loss at an optimal configuration.

Categories and Subject Descriptors
C.1.3 [**Processor Architectures**]: Other Architecture Styles – *pipeline processors.*

General Terms
Measurement, Performance, Design, and Experimentation,

Keywords
Address compression, Load-store queue, Power-efficient, Scalable design.

1. INTRODUCTION
Embedded systems running operating system and being capable of multi-tasking have become more popular in recent year. The increasing demand on computation power has pushed the mainstream manufacturer such as ARM to promote superscalar cores like Cortex-R4. In a contemporary superscalar processor, the load-store queue (LSQ) is integrated into memory stage for detection and resolution of access violation and ordering issues. However when the size of LSQ increases as the issue width of superscalar processor become wider, the CAM-based LSQ structure will face scalability issue of both power and area. It seems inevitable to use large memory matrices to handle these addresses, but as the timing requirement is pushed more strictly the power and area cost of these storage-based components become more critical. Since the power and area are both proportional to size of memory, reducing the size of memory component can almost directly translate to lower power and area requirement. Compression techniques have been widely applied to memory systems storing programs and data to reduce the required size or expand the bus bandwidth. Complex or simple, these literatures are all based on re-encoding the repeated contents of the memory to store them in a more concise way. The reason why compression techniques are applicable to store code and data also holds to address. Figure 1 shows the user space arrangement of the MIPS32 platform. As we can see in the figure, the compiler statically binds the address into the different memory segments so that data belonging to the same segment inherently have the same base address and present strong spatial locality.

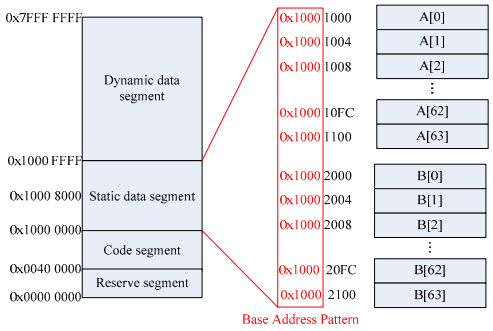

Figure 1. Data arrangement in user space of MIPS32

Due to the strong spatial locality, address is a good subject to apply compression techniques. Note that the spatial locality of address mentioned above exists in all systems but for systems with virtual address management capability, the locality is even stronger thus beneficial to apply compression. Table 1 shows the different pattern counts of the leading 16 bits (16B) and leading 20 bits (20B) of the data virtual addresses collected from selected programs in SPEC2000[9]. As observed in Table 1, most programs have limited number of patterns compared to the total memory instruction count, thus a simple re-encoding process of these massively repeated patterns is anticipated to have a positive effect on scaling the size of LSQ.

In this paper, we propose a dynamic virtual address compression scheme to improve the scalability of LSQ. We improved the microarchitecture of LSQ to enable the processor to manage addresses in compressed form and evaluate the power and area

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

reduction ratio of the proposed design. The effectiveness of different compression parameters is also presented.

Table 1. Pattern counts of the data addresses

SPECint	gzip	place	route	gcc	mcf	paser	bzip2		
instruction count	1.18G	2.19G	1.12G	959M	115M	1.69G	3.89G		
16B	110	6	69	2303	1483	642	1491		
20B	1628	52	430	9835	23694	2608	9802		
SPECfp	wupwise	swim	mgrid	applu	mesa	art	equake	ammp	apsi
instruction count	12.4G	311M	13.6G	223M	1.02G	689M	561M	2.18G	4.58G
16B	2817	1188	895	158	149	34	158	339	3056
20B	>40000	14382	14201	899	2299	504	2484	5368	>40000

The remainder of this paper is structured as follows. Section 2 presents our survey of the related work. Section 3 describes the proposed address compression methods. Section 4 discusses the pipeline hazard issues induced by address compression and their solutions. Then the implementation details and hardware models for estimating area and power are described in Section 5. Section 6 presents the results of our experiment and analysis of the influence on area cost, energy consumption and performance. Finally, in Section 7 we conclude and summarize this paper

2. RELATED WORK

Address compression has been used in system bus to reduce wires connecting components thus lower the energy dissipation or relax the routing restrictions. The base register caching [1] was applied to system bus to compress addresses. The high-order bits of addresses are replaced with a shorter index to a base register. The base register indexes are then used to transmit addresses across the narrowed system bus. The project BUS-EXPANDER [2] extends the compression target of [1] to data bus. In [10] a partial match mechanism is applied to improve the hit rate of the base register cache and attain better performance. These researches mainly focus on physical address compression and are applied to buses of lower memory hierarchy where timing constraints are much more relaxed than processor core.

Among the LSQ design issues, the published literature can be divided into two major categories. One is to reduce the searching times of the LSQ, [8][14] using a table to record and predict the memory collision when load-store pair occurs. The modification on pipeline structure to avoid collision check in LSQ [7] is also proposed to reduce the dynamic power of LSQ. The other category is to apply set-associative structures to LSQ to reduce the search range and latency [11][13]. Castro et al. [5] proposed a hybrid scheme of the above which modifies the load queue and integrates a table similar to [14] to reduce collision check times.

Although address compression and LSQ energy-efficient issues have been investigated, there's no research project, to our knowledge, combining the address compression and LSQ energy-efficient topics as we do. Our work further proves the feasibility and effectiveness of compressing virtual addresses in a superscalar processor where timing and power demands are more stringent.

3. VIRTUAL ADDRESS COMPRESSION

In the following, we present the technique of applying virtual address compression on LSQ. For simplicity, the term "address" refers to virtual address in all subsequent paragraphs. Our proposal assumes that the target system is equipped with virtual memory management capability so that all addresses flow inside the pipeline are virtual addresses.

The basic concept of compressing addresses using dictionary-based algorithm is similar to the idea proposed in previous works [1][2] but we further refine it to handle the complex hazard conditions in the pipeline. The compression and decompression mechanisms are integrated into one lookup table and adopted into the memory scheduling stage. We call this table Data Address Pattern Table (DAPT) and its relation to other components is depicted in Figure 2.

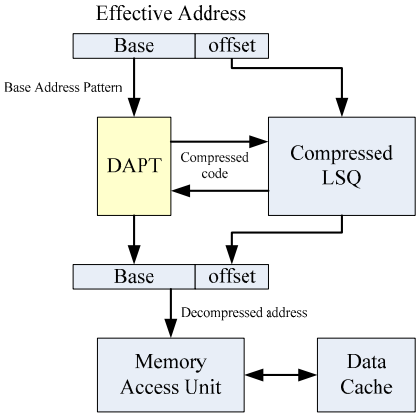

Figure 2. Applying address compression to LSQ

As shown in Figure 2 once the effective address of a load or store instruction is generated, its high-order bits, i.e. the base address, are compressed by DAPT before entering LSQ. The DAPT is a CAM-based lookup table storing the mapping relationships of base address pattern and its index code. A lookup process is performed for each base address pattern and if it hits the DAPT, its corresponding code is sent to LSQ. If no matched pattern is found, an empty entry inside DAPT will be selected to store the address. When there is no entry available, the compression process for the incoming pattern will be deferred and the memory scheduling pipeline will be stalled until a DAPT entry is available. The compressed LSQ stores the code of compressed base address along with the uncompressed offset. When the memory scheduler circuit decides one or more entries of LSQ are to be issued to the memory access unit, the DAPT will be indexed using the code to readout the base address.

Because of the first-in-first-out (FIFO) nature of LSQ, the DAPT can employ a simple replacement mechanism to reduce the hardware complexity. Once an instruction is committed and leaves the LSQ, its corresponding address pattern can be cleared from the DAPT if no other base address in LSQ has the same pattern. Since all memory instructions go through the LSQ sequentially, we can attach a counter to each entry of DAPT and increase the counter when a new address with the same base pattern enters the LSQ. Whenever an instruction leaves LSQ, its corresponding pattern counter is decreamented and if the counter reaches zero, that entry is invalidated and becomes a candidate for DAPT replacement.

4. PIPELINE HAZARD ISSUES

The counter-based table updating along with the deferred compression mechanism contribute to simplify the hardware complexity of the DAPT but will impact performance and generate two hazards for the pipeline. Under certain conditions the processor will be locked up because a resource deadlock could occur in the DAPT. Another hazard occurs when an exception or speculation failure forces the processor to rollback the pipeline to an earlier state. To deal with these hazards we devised two procedure namely DAPT bypassing and DAPT recovering. The following sub-sections will discuss why the hazards occur and how we solve them.

4.1 DAPT Bypassing

The deferred compression will cause a deadlock if the incoming instruction happens to be at the head of the reorder buffer (ROB). The ROB holds the in-flight instructions and commits them in program order when the result of the instruction is ready. Since the head entry of the ROB holds the most urgent instruction to be committed, no other instruction can leave the pipeline before the head entry is committed. If such an urgent instruction is dispatched to the LSQ and the DAPT is full, the deadlock occurs because the incoming instruction is deferred and no other instruction in the LSQ can be committed to reset the counters in DAPT.

The solution for this problem is to bypass the urgent instruction, i.e., do not schedule the instruction into the LSQ, and hence the urgent instruction will be processed directly without being queued. After the urgent instruction is committed, the remaining instructions in the LSQ will be able to update the DAPT and the deadlock is avoided.

4.2 DAPT Recovering

When a branch miss-prediction or exception happens, the pipeline needs to flush instructions fetched after the exception is signaled. If the instructions which are being flushed include memory access ones, the LSQ will also need to recover to an earlier state i.e. flush some entries in it. However, it's essential that we keep the counters in DAPT updating correctly, so any instruction being committed or flushed out of the LSQ should have its corresponding counter in the DAPT updated.

To properly update the DAPT counters after the LSQ flush operation, a hardware buffer called recover list (shown in Figure 3) is integrated into the DAPT. Moreover, a counter-valid bit is appended to each entry of the LSQ to indicate whether the address pattern of the entry is still used by the DAPT. Whenever an address is compressed and enters the LSQ, the counter-valid bit is set to 1 and when it's committed with proper updating to the DAPT counter, the bit is reset to zero. A LSQ flush operation caused by an exception won't reset the counter-valid bit to 0 so we can tell whether the entry is flushed or committed.

After an exception has occurred, there might be some instructions that are flushed in the LSQ. The entries occupied by the flushed instructions will be overwritten by the newly arriving ones. If an LSQ entry with counter-valid bit set to 1 is being overwritten, then the pattern code of the entry will be copied to the Recover List before the new value replaces it. This copy operation may increase the latency of memory scheduling pipeline but will ensure the DAPT counter to be updated correctly. The Recover List will be checked each cycle and if it holds valid code, the corresponding counter in the DAPT will be decreased. The code will be removed from the Recover List once it has completed updating of the DAPT counter.

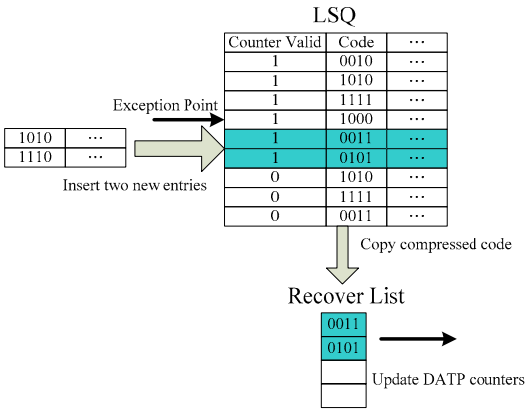

Figure 3. Recovering compressed code after exception

5. IMPLEMENTATION ISSUES

This section presents the implementation issues including the hardware and power model we used to estimate the area and energy usage of the proposed design. The models discussed in this section will be used to derive various data of different configurations of the proposed design.

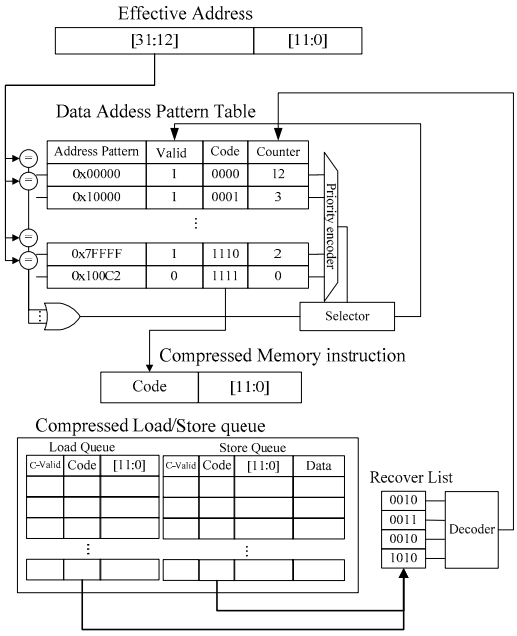

Figure 4 Circuit diagram of Data Address Pattern Table and compressed LSQ

Figure 4 shows the detailed circuit diagram of DAPT and compressed LSQ. In this exemplary embodiment, the higher 20 bits of the base address are compressed by DAPT and the contents of LSQ entries are modified accordingly. The DAPT is basically a CAM-based storage component with counter appended to each entry for correct replacement as mentioned in section 4. The 4-bit

code length implies that the entry count of DAPT in this example is 16 and in later sections we name such configuration as "16E20B." We assume that the target platform uses separated load and store queues to attain better area yield in exchange of the flexibility of an unified LSQ.

As we can see in Figure 4, applying address compression helps shrinking the area of LSQ by reducing the recorded address length in each entry. Since the area is dominated by the large amount of storage component, we estimated total area by calculating the equivalent RAM cell area of each storage component and accumulating them. Note that we assume both the address pattern field of the DAPT and address field of LSQ are implemented with CAM cells. A popular rule of thumb is to count each CAM cell as two RAM cells [12], so the area approximation of the DAPT and compressed LSQ in the unit of bit are given by the following formulas:

$Area^{DAPT}$ =

Entry_count ×[Fan_in × (2×Address_Pattern_width) + Valid_bit + Counter_width] + Recover_list_size

$Area^{LSQ}$ =

Entry_count × { 2× [Fan_in × 2× (Code_width + offset_width) + Counter-Valid_bit] + Data_width **}**

As for the power model, we use Wattch [4] platform to estimate the dynamic power usage of the related component. To provide a more precise processor-wide dynamic power accumulation, we use the implementation parameters derived from CACTI 3.0 [6] instead of the original version embedded in Wattch. The derived parameters are used to estimate the power of RAM matrices inside caches and branch target buffer. In contrast with RAM cells, the DAPT and LSQ are mainly based on CAM structure so we update the CAM dimension parameters of power estimation subroutine in Wattch to fit the proposed design. Thus the modified Wattch can faithfully estimate the power consumption of compressed LSQ and DAPT.

6. SIMULATION RESULT AND ANALYSIS

This section presents various experimental results of different compression parameters to evaluate the effectiveness of address compression. We modified the architectural power estimation framework Wattch [4] which is based on SimpleScalar [3] to simulate the behavior of the proposed design. A set of selected programs from SPEC2000 [9] benchmark suite listed in Table 1 are used. The baseline system configuration is listed in Table 2. We set the total entry counts of LSQ to half that of the ROB because memory instructions occupy about 30-45% of dynamic instruction counts. The simulation is performed by fast-forwarding the initial 100 million instructions and collect statistics from the succeeding 500 million instructions. The performance (and energy) of the processor with this configuration is used as a normalization baseline to other systems with various compression parameters.

Table 2. Baseline system configuration

Issue width	4 inst / cycle
ROB size	256 entries
LQ-SQ size	64-64 entries
Functional unit	4 INT ALU , 1 INT mult / div , 2 FP Adders , 1 FP mult /div
Branch predictor	Bimodal : 2K entries
BTB	1K entries / 4-way / 256-set
L1 data cache	64KB 4-way
L1 instruction cache	64KB 4-way
L2 cache	256KB 4-way

Several configurations of DAPT are selected for simulation to demonstrate their effects on area and energy reduction. In Table 3, the configuration labeled 8E16B means that the DAPT has 8 entries and the compressed base address is 16-bits wide. These parameters are chosen based on the area and power estimation to keep the cost of DAPT in reasonable range. The results are shown in the following sub-sections and analyses of the data are given.

Table 3. Configurations of DAPT

Compression configurations	
DAPT in LSQ	8E16B / 8E20B / 8E24B / 8E28B
	16E20B/16E24B/16E28B/16E32B

6.1 Scaling Area and Power

Figure 5 shows the power consumption and area usage after applying address compression to the LSQ. For each benchmark program, both the energy consumption of the compressed LSQ and DAPT are accumulated throughout the program execution. The area estimation formula described in previous section is used to derive the area usage of each configuration. Both power and area are normalized to the baseline system to show the scaling effect. The result of an additional configuration labeled L32-S32, which represents a baseline system with half-sized LSQ, is also provided as reference.

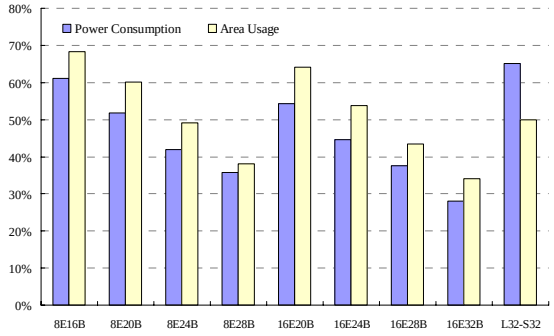

Figure 5. LSQ Energy consumption and area usage rate

As we anticipated, the compression of LSQ has positive effect on scaling down the dynamic energy consumption and cost. The compressed LSQ can reduce power consumption ranging from 38% to 71% depending on the compression width. The reduction rate is proportional to the compressed pattern width and disproportional to the entry count of DAPT. The configurations which compress more than 24 bits can substantially scale down the power and area even better than a half-sized LSQ. It's obvious that compression helps to reduce the CAM-based area of the LSQ, thus lower the dynamic power dissipation of the component.

6.2 Processor-wide Energy and Performance

The overall processor IPC and energy consumption is shown in Figure 6. We collocate the IPC ratio with total energy consumption normalized to baseline system so that the balance between performance and energy consumption is emphasized. The configuration which saves most component power and area in Figure 5 does not give good IPC and thus results in increased processor energy consumption. For a processor-wide point of view, the best configuration to use is 16E20B and 16E24B. These 2 configurations can reduce about 3.08% processor-wide energy consumption with 0.22% IPC degradation. And from Figure 5 we can tell only less than 65% area of a conventional LSQ is required for both configurations.

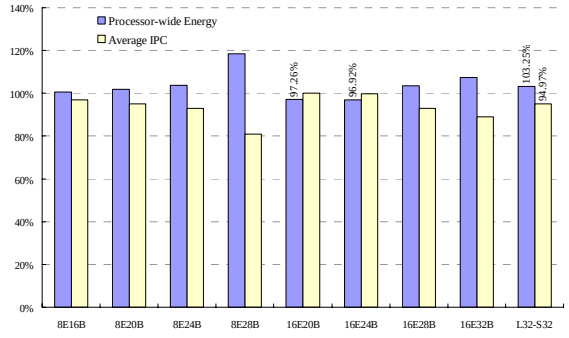

Figure 6. IPC and processor energy consumption rate of different compression parameters

The comparison to L32-S32 configuration further proves our hypothesis on the scaling effect of applying address compression. Given appropriate configurations of DAPT, the compressed LSQ can scale down the area and power requirements close to a half-sized conventional LSQ while providing better IPC performance.

7. CONCLUSION

This paper presents address compression scheme to scale down the power and area cost of a LSQ in contemporary superscalar processors. A table-based compression circuit for the LSQ called DAPT is proposed. Various compression parameters are applied to the proposed scheme and the improvement of scalability is evaluated in the aspect of area reduction, energy consumption, and performance impact. Our experiment demonstrates a reduction on the area cost ranging from 32% to 66% and energy consumption ranging from 39% to 72% in different configurations. The impact to IPC and processor-wide energy consumption are also presented. With appropriate compression parameter, the proposed scheme can scale the existing LSQ design to about half area and provide 3% processor power reduction with negligible IPC loss.

8. ACKNOWLEDGMENT

This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC 96-2221-E-006-192-MY3 and by the Program for Promoting Academic Excellence of Universities in Taiwan.

9. REFERENCES

[1] A. Park and M. K. Farrens, "Address Compression through Base Register Caching," in *Proceedings of the Annul IEEE/ACM International Symposium on Microarchitecture*,1990 , pp.193-199.

[2] D. Citron and L. Rudolph, "Creating a Wider Bus Using Caching Techniques," in *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, 1995, pp.90-99.

[3] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0", Technical Report CS1342, University of Wisconsin-Madison, Jun. 1997.

[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000, pp.83-94.

[5] F. Castro, D. Chaver, L. Pinuel, M. Prieto, M .C. Huang, and F. Tirado, "LSQ: a power efficient and scalable implementation," in *IEE proceedings on Computers and digital Techniques*, 2006, pp.389-398.

[6] G. Reinman and N. P. Jouppi, "CACTI 3.0: An Integrated Cache Timing and Power Model," Technical Report, COMPAQ Western Research Lab, Palo Alto, CA, Aug. 2001.

[7] H. W. Cain and M. H. Lipasti, "Memory Ordering: A Value-Based Approach," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004, pp.90-101.

[8] I. Park, C. L. Ooi, and T. N. Vijaykumar, "Reducing Design Complexity of the Load/Store Queue," in *Proceedings of the 36th Annul IEEE/ACM International Symposium on Microarchitecture*, 2003, pp.411-422.

[9] J. L. Henning, "SPEC CPU2000: Measuring CPU performance in the new millennium," *IEEE Computer*, Vol: 33, 2000, pp.28-35.

[10] J. Liu, K. Sundaresan, and N. R. Mahapatra , "A Fast Dynamic Compression scheme for Low-Latency On-Chip Address Buses," *Proceedings of the 4th Workshop on Memory Performance Issues*, 2006.

[11] J. Abella and A. González, "SAMIE-LSQ: Set-Associative Multiple-Instruction Entry Load/Store Queue," in 20th *IEEE International Parallel and Distributed Processing Symposium*, 2006.

[12] Kostas Pagiamtzis, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," in *IEEE Journal of Solid-State Circuits*, 2006, pp.712-727.

[13] L. Baugh and C. Zilles, "Decomposing the Load-Store Queue by Function for Power Reduction and Scalability," in *IBM Journal of Research and Development in Computers & Technology*, 2006, pp.287- 297.

[14] S. Sethumadhavan, R. Desikan, D. Burger. C. R. Moore, and S. W. Keckler, "Scalable Hardware Memory Disambiguation for High ILP Processors," in *Proceedings of the 36th Annul IEEE/ACM International Symposium on Microarchitecture*, 2003, pp.188-127.