Address Compression for Scalable Load/Store Queue Implementation

Yi-Ying Tsai, Chia-Jung Hsu, and Chung-Ho Chen
Department of Electrical Engineering and
Institute of Computer and Communication Engineering, National Cheng Kung University
Tainan, Taiwan
{magi,jovi}@casmail.ee.ncku.edu.tw, chchen@mail.ncku.edu.tw

***Abstract*—Contemporary superscalar processors employ the load/store queue for memory disambiguation. A load/store queue is typically implemented with a CAM structure to search the address for collision and consequently poses scalability challenges of energy consumption and area cost. This paper proposes an address compression technique for load/store queue to improve the power efficiency and scalability. Using the proposed approach, the LSQ can reduce the energy consumption ranging from 38% to 72% and area cost ranging from 32% to 66%, depending on the compression parameter and system configuration. The approach can provide 3.08% overall processor energy reduction and causes only 0.22% performance loss at a balanced configuration.**

I. INTRODUCTION

In contemporary superscalar processors, the load-store queue (LSQ) is integrated into memory stage for the detection and resolution of access violation and ordering issues. However, when the size of the LSQ increases as the issue width of superscalar processor is increased, 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.

Compression techniques have been widely applied to memory systems storing programs and data to lower the required size and power. The reason why compression techniques are applicable to code and data also holds to address. Fig. 1 shows an example of address distribution of two arrays. As we can see in the figure, the compiler statically binds the address of code and data into different memory segments so that data belonging to the same segment inherently have the same base address and present strong spatial locality in their address patterns. Due to such 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 since the relocation process is handled by the MMU and transparent to the processor itself.

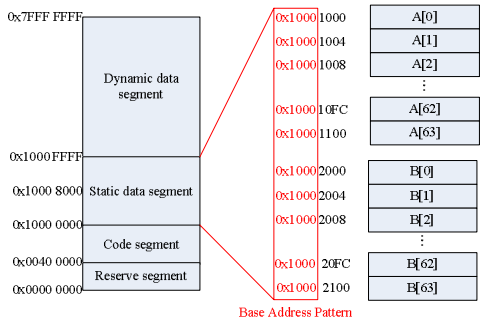

Figure 1. Spatial locality of base address pattern

In this paper, we propose a dynamic address compression scheme to improve the power efficiency and scalability of the LSQ. We modified the microarchitecture of the LSQ to enable the processor to manage virtual addresses in compressed form. The power and area reduction ratios of the proposed design are evaluated with a cycle-accurate performance/power simulator. The simulation results of different compression parameters are presented and discussed.

II. ADDRESS COMPRESSION

The basic concept of compressing addresses using dictionary-based algorithm is similar to the idea proposed in [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 LSQ. We call this table Data Address Pattern Table (DAPT) and its relation to other components is depicted in Fig. 2.

As shown in Fig. 2, once the effective address of a load or store instruction is generated, the high-order bits, i.e. the base address, are compressed by the DAPT before entering the LSQ. The DAPT is a CAM-based look-up 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, the corresponding code is sent

978-1-4244-1684-4/08/$25.00 ©2008 IEEE

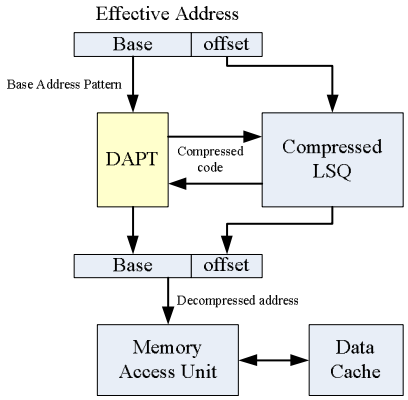

Figure 2. Applying address compression to LSQ

to the 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, thus lower the required CAM-based storage. When the memory scheduler circuit decides one or more entries of the LSQ are to be issued for memory access, the DAPT again will be indexed using the code to readout the base address.

The DAPT replacement is a simple counter-based mechanism. A counter is attached per entry of DAPT to indicate how many entries in the LSQ are encoded using that pattern code. Every address entering the LSQ will increment the corresponding counter in DAPT. Whenever an address leaves the LSQ, its corresponding pattern counter is decreased by one and if the counter reaches zero, that entry is invalidated and becomes a candidate for DAPT replacement. Such counter-based replacement has side effect of generating additional two pipeline hazards and are discussed in the next section.

III. IMPLEMENTATION ISSUES

In this section several implementation issues are discussed. First, the cost of DAPT must be kept at a reasonable range so that it can benefit the overall power and area usage of the compressed LSQ. Second, the counter-based table updating along with the deferred compression mechanism contributes to simplify the hardware complexity of the DAPT but will impact performance and generate two hazards for the pipeline. The following sub-sections will discuss why these issues occur and how we solve them.

A. Cost of DAPT

Fig. 3 shows the detailed circuit diagram of the DAPT and the compressed LSQ. In this exemplary embodiment, the higher 20 bits of the base address are compressed by the DAPT and the contents of the LSQ entries are modified accordingly. The DAPT is a CAM-based storage component with a counter appended to each entry for correct replacement as mentioned above. The 4-bit code length implies that the

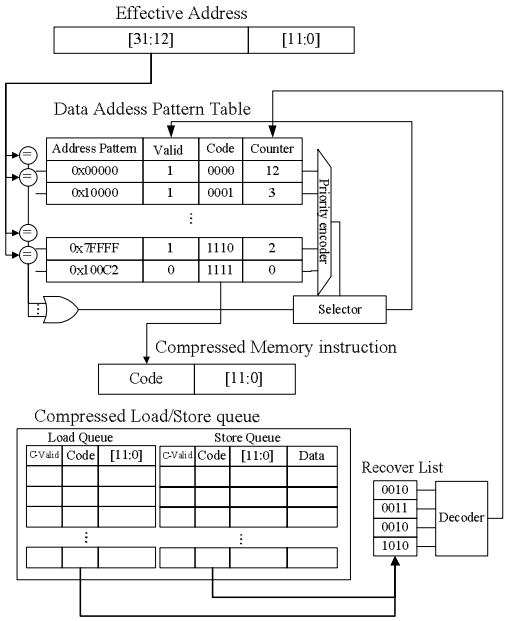

Figure 3. Exemplary Embodiment of DAPT and compressed LSQ

entry count of the DAPT in this example is 16 and in later sections we name such configuration as "16E20B."

It's obvious the wider the compressed address is, the more area and power can be saved in the LSQ. However, expanding the width of compressed address will contradictorily weaken the spatial locality therefore place more pressure on DAPT replacement. And the DAPT replacement pressure can be mitigated by increasing the entry count. Since the cost of DAPT is proportional to both entry count and compressing width, it would be favorable if a balance between spatial locality and DAPT compression efficiency can be found. From preliminary statistics of program behavior and estimation of area usage, we concluded that at least 16 bits of the address should be compressed and the DAPT should contain more than 8 entries. The effects of various compression parameters on energy consumption, area and performance are presented and discussed in Section V.

B. Deadlock in DAPT

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 make the LSQ 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. Such

modification to pipeline will degrade performance since the bypassed load or store won't have chance to forward or squash redundant memory instructions in the LSQ. However, our experiment shows the impact on performance lies in an acceptable range.

C. Incomplete DAPT Counter Updating

When a branch misprediction 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, the flushed load or store instruction will cause the incompleteness of DAPT counter updating. If the flushed memory instructions are just discarded, the corresponding address pattern counters will never reach zero and become resistant to DAPT replacement. Hence it's essential for any instruction being committed or flushed out of the LSQ to 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 Fig. 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 counted by the DAPT. Whenever an address is compressed and enters the LSQ, the counter-valid bit is set to one 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 zero so we can tell whether the entry is flushed or committed.

If a LSQ entry with counter-valid bit set to 1 is being discarded, then the pattern code of that 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:

IV. Experimental Platform

We modified the architectural power estimation framework Wattch [4] and SimpleScalar [3] to simulate the behavior of the proposed design. To provide a more precise processor-wide dynamic power accumulation, we use the implementation parameters derived from CACTI 3.0 [5] instead of the original version embedded in Wattch. Since the DAPT and LSQ are mainly based on CAM structure, we also update the CAM dimension parameters of power estimation subroutine in Wattch to fit the proposed design so that the modified simulator can faithfully estimate the energy consumption of compressed LSQ and DAPT.

The baseline system configurations and proposed address-compressing scheme with various DAPT parameters are listed in Table I. We set the baseline configurations to model high issue rate and large instruction window with different LSQ capacity. The selected applications from SPEC2000 [6] benchmark suite are also listed in Table I. 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.

TABLE I. Simulation Configurations

Processor	
Issue Width	4-issue out-of-order processor
ROB size	256 entries
Functional units	4 INT ALU, 1 INT Mult/Div, 2 FP Adder, 1 FP Mult/Div
Branch Prediction	Bimodal 2k entries, 1k BTB
L1 data cache	4-way 64KB
L1 instruction cache	4-way 64KB
L2 unified cache	4-way 256 KB
Memory latency	75 cycles
Load / Store Queue Sizes	
Baseline #1 L32-S32	32/32 entries
Baseline #2 L64-S64	64/64 entries
DAPT parameters	
8-entry compression width	16 bits, 20 bits, 24 bits, 28 bits (8E16B,8E20B,8E24B,8E28B)
16-entry compression width	20 bits, 24 bits, 28 bits, 32 bits (16E20B, 16E24B, 16E28B, 16E32B)
Benchmarks	
SPECint	gzip, vpr.place, vpr.route, gcc, mcf, parser, bzip2
SPECfp	wupwise, swim, mgrid, applu, mesa, art, equake, ammp, apsi

V. Result and Analysis

This section presents various experimental results of different compression parameters. The area cost of each variant scheme and the baseline system are also estimated to emphasize the scalability issue. Since the area of LSQ 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 the LSQ are implemented with CAM cells. A popular rule of thumb is to count each CAM cell as two RAM cells [7]. The improvement of power efficiency is measured and presented in the form of energy consumption ratio.

A. Energy Consumption and Area Recuction

Fig. 4 and Fig. 5 show the energy consumption and area usage after applying address compression to the systems with different LSQ size. Both energy consumption and area are normalized to the baseline systems to show the scaling effect. As we anticipated, the compression of LSQ has positive effect on scaling down the dynamic energy consumption and area. The compressed LSQ can reduce energy consumption ranging from 39% to 72% for L32-S32 and 38% to 71% for L64-S64, respectively. The reduction rate is proportional to the compressed pattern width. Comparing Fig. 4 and Fig. 5, we can tell that the larger the LSQ is, the more evident the area scaling effect becomes.

In Fig. 5, the energy consumption and area of the baseline system L32-S32 are also appended to show that compressing 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 mitigate the scalability issue as the size of the LSQ grows.

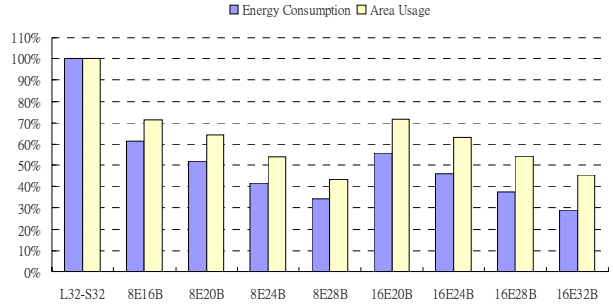

Figure 4. Applying address compression to baseline #1

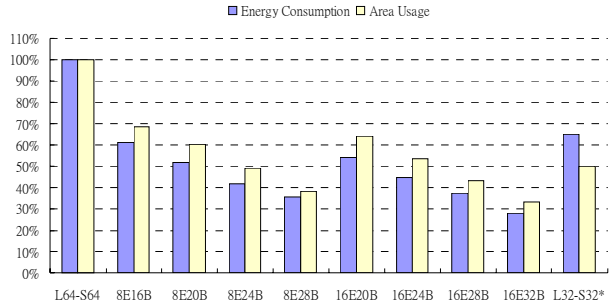

Figure 5. Applying address compression to baseline #2

B. Processor-wide Energy Consumption and Performance

The average IPC and processor energy consumption are shown in Fig. 6 and Fig. 7, respectively. We show the IPC and total energy consumption both normalized to the baseline systems so that the balance between performance and energy consumption is emphasized. Increasing the entry count of DAPT helps mitigate the impact on performance. From a processor-wide point of view, the preferable configuration is 16E24B. This configuration can reduce processor-wide energy consumption up to 3.08% with negligible (0.22%) IPC degradation when applied to the L64-S64 system.

VI. CONCLUSION

This paper presents schemes that integrate address compression technique into 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 38% to 72% in different configurations. The impact to IPC and processor-wide energy consumption are also presented. With appropriate compression parameter (16E24B), the proposed scheme can scale the conventional LSQ design to less than 60% area and provide average 3% processor power reduction with negligible IPC loss.

As current processor design technology has migrated to 64-bit systems, the scalability challenges of energy and area originate from wide address bus will emerge. Since the spatial

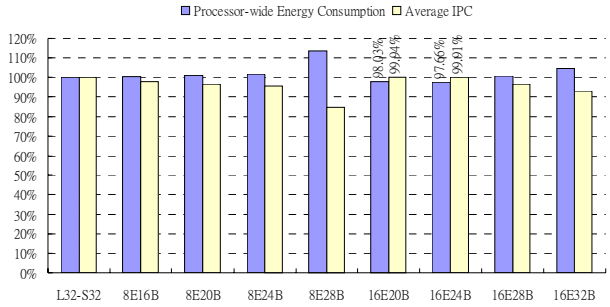

Figure 6. Processor-wide energy consumption and performance after applying address compression to baseline#1

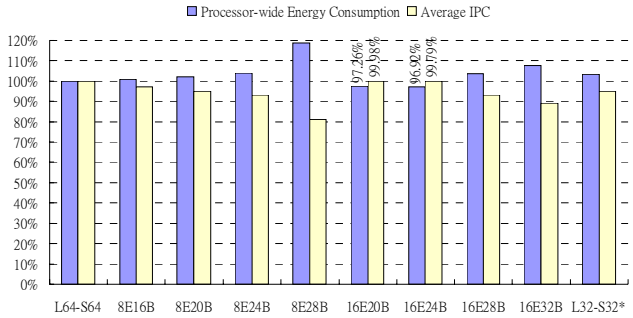

Figure 7. Processor-wide energy consumption and performance after applying address compression to baseline #2

locality of address will continue to exist and become more obvious in the large address space, there are greater chances to benefit from compressing the wide address bits.

ACKNOWLEDGMENT

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

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] 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.

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

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