

# Design Trade-offs for Load/Store Buffers in Embedded Processing Environments

Young Hoon Kang

Ming Hsieh Department of Electrical Engineering  
University of Southern California  
Los Angeles, CA/USA  
youngkan@usc.edu

Jeffrey Draper

Information Sciences Institute  
University of Southern California  
Marina del Rey, CA/USA  
draper@ISI.EDU

**Abstract**— Memory latency is a critical issue for conventional high-speed computing platforms, and it is becoming a common problem in embedded and CMP (chip multiprocessing) systems as well. Conventional processors typically adopt caches and a Load/Store Queue (LSQ) to address the processor-to-memory bottleneck. However, the conventional LSQ design, which has a large number of entries, is not appropriate for embedded systems due to its area and power hungry out-of-order speculation. A compact, low-power load/store buffer that also provides significant performance improvement is essential for such systems. In this paper, we propose an area-efficient WideWord Load/Store Buffer (WLSB) which supports both WideWord (256-bit) and scalar (32-bit) load/store instructions for a recently fabricated PIM (Processing-In-Memory) device [1]. Given its small size, the 4 entry WLSB yields a 57.33% load hit rate on SPEC2K [3] benchmarks. This result is 5.72% better as compared to a less area-efficient 32-entry fully associative scalar load/store buffer (SLSB). The WLSB was synthesized in IBM 90 nm technology, and the resulting implementation occupies less than a seventh of a square mm and is projected to run at 1.6ns cycle time with 15.72mW of dynamic power dissipation. This paper demonstrates how this very small-entry buffer can affect the load hit rate and quantifies the design trade-offs between wide small-entry and narrow large-entry buffers with respect to size, power, load hit ratio and clock speed. Although this WLSB has been specifically designed to benefit a PIM architecture, it is expected to be useful for other embedded processing platforms and CMPs due to emphasized area/power constraints.

## I. INTRODUCTION

Processing-in-memory (PIM) [1] has been proposed as a solution to the memory wall problem. It yields dramatically increased memory bandwidth by the inherent nature of an embedded processor directly connected to a memory bank. Although processing-in-memory architectures like the Data-Intensive Architecture (DIVA) [1] have significant memory latency advantages over conventional systems, as fabrication technologies advance, latency to on-chip embedded DRAM (eDRAM) is increasing. Conventional systems have employed data caches and load/store queues (LSQ) to combat increasing latency. Given the area and power constraints of PIM systems and similar parallel embedded systems, data caches are not deemed feasible, especially considering cache coherence issues that require even more

hardware resources. However, a small load/store buffer has great potential for accelerating PIM performance by decreasing data access times for load/store instructions. This paper introduces the WideWord load/Store Buffer (WLSB) which supports both instruction types for the DIVA PIM and resolves the memory aliasing problem caused by two different types of instruction sets. The proposed WLSB is sufficiently fast and yields a significant performance increase with little hardware costs. Given its small size, the proposed 4-entry WLSB also has a faster access time as compared to a 32-entry fully associative scalar load/store buffer (SLSB), which has the same number of data register bits. As synthesized in IBM 90nm technology, the proposed WLSB is 37.5% faster and occupies 36% less area than the 32-entry fully associative SLSB. In addition, the WLSB improves the load hit rate by 5.72% for 11 benchmarks in the SPEC CPU2K [3] suite and 24.73% for 4 SPEC FP2K benchmarks. The increased performance on the SPEC FP2K benchmarks especially demonstrates how the longer line size of the WLSB benefits applications with spatial locality.

In Section II, the WLSB structure and operation are explained, and in Section III, several replacement algorithms applied to WLSB are proposed. Section IV discusses simulation, implementation, and evaluation. Section V concludes the paper.

## II. WIDEWORD LOAD/STORE BUFFER

Fig. 1 shows a depiction of the WLSB and its placement in the memory stage of a pipelined processor. There are 4 or 8 comparators for a fully associative search when the number of entries is 4 or 8, respectively. Each entry of the WLSB consists of a 27-bit WideWord address, 8 store valid bits, 8 cache valid bits and a 256-bit data field. One entry corresponds to either a single WideWord memory instruction or up to 8 scalar instructions of 8 consecutive memory addresses. Within the 32-bit addresses, the least significant 2 bits are ignored since all access are aligned to a 4-byte boundary and the next 3 bits are decoded to specify which 32-bit word is being accessed within the 256-bit data field. The store valid field indicates that the store instruction is stored in the entry but is still not committed to memory, whereas the cache valid field shows that the slot or the entire entry is valid and consistent with the corresponding memory

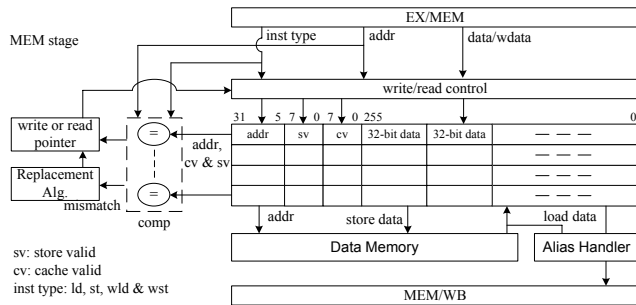


Figure 1. WideWord Load/Store Buffer

location.

If a store instruction is issued to the memory stage, the store valid bit is set in the slot of the entry in which data is placed. When the store instruction commits to memory, the store valid bit is reset and the cache valid bit of the same slot and entry is set while still maintaining the data. Thus, the WLSB exhibits caching behavior. After a load/store instruction commits to memory, data is still retained in the WLSB for later reuse. If a new store instruction needs to replace an entry, the corresponding cache flag bit is cleared, and the store valid bit is set. The data field is only updated when the entry replacement is requested by other memory instructions or a new store instruction tries to update a matching entry. All these operations are done through a write/read control unit that fetches the pointer of a matched entry or a replacement in case of a mismatch.

Each load instruction triggers one of two operations. If a matched entry is found based on the address and two valid fields, cache and store, store-to-load forwarding is possible. Subsequent load instructions can be combined with prior load instructions that map to the same address of an entry in the WLSB for which the memory load has already been initiated or completed. This greatly helps to reduce memory traffic. When the load instruction misses the WLSB, the data are fetched from memory and placed in one of the entries, and the 8 cache valid bits are all set since the fetched data from memory is always 256-bit. However, the two different types of instructions may result in a RAW hazard. Fig. 2 shows an example of the RAW hazard. While a scalar store is in the WLSB, if a subsequent WideWord load with the same address ‘A’ occurs in the memory stage, it must take into account the scalar store. The WideWord load directly accesses the memory because only the partial 32-bit data is available in the WLSB. This process causes a RAW hazard. The WLSB resolves this problem by merging the fetched 256-bit data with the value of the in-flight store instruction in the entry. Any other memory hazards like WAW and WAR are not possible in this DIVA PIM processor because it is a single-issue and in-order execution processor.

With the features of store-to-load forwarding and caching behavior so that repeated stores to the same location take place in the WLSB rather than accessing memory, the WLSB greatly reduces memory traffic. The WLSB commits store instructions to memory in FIFO order. A committed

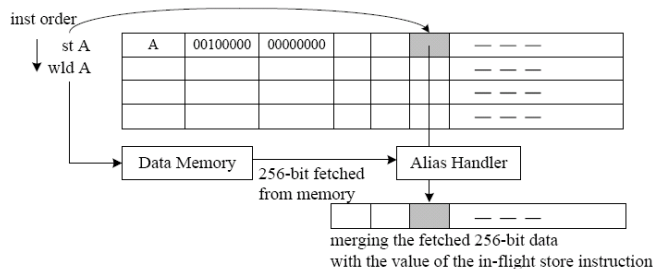


Figure 2. Merging 256-bit load data with in-flight store 32-bit data

store and fetched load entry can be evicted any time as needed by a more recent load or store instruction for replacing. Replacement is based on the algorithms described in Section III. If all the entries are occupied by store instructions with the store valid bits and another store is ready in the memory stage, the pipeline is stalled until one of the entries is committed.

### III. REPLACEMENT ALGORITHMS

The WLSB is inherently designed to easily fit into fast and light embedded processors. It should not impose much overhead to the processor in terms of area and power, and not affect the processor clock frequency. This suggests that some complex and long-latency replacement algorithm which guarantees a high load hit rate cannot be applied to the WLSB. In this paper, we introduce the following 3 simple replacement algorithms which maintain an appreciable load hit rate.

#### A. FIFO

A naïve FIFO replacement has been tested as a baseline algorithm to compare with other results. Entries are replaced in FIFO order but a pending store instruction entry cannot be a candidate for replacement. This rule also applies to all other replacement algorithms.

#### B. 1 Dedicated load

This algorithm reserves at least 1 entry for load instruction data. For this algorithm, one bit “ld/st” field has been added to classify instructions. This replacement algorithm started from the consideration that load instructions might have higher hit probability than store instructions. To fully utilize the WLSB, it is better to retain as many load instructions as possible because a non-matched scalar store instruction occupies only one 32-bit word of a 256-bit data field.

#### C. Order revision next to the hit point

This algorithm replaces the entries in a FIFO order, but if it encounters any matched entry, then the next replacement point is changed to right after the matching entry. This algorithm attempts to provide a longer lifetime to frequently matched entries. Fig. 3 shows an example. The instructions are part of the art benchmark from SPEC2K. Several memory instructions with the same address are issued

repeatedly. Both the FIFO and Hitpoint replacement algorithms work in the same way until the 7th instruction. However, after the 8th instruction, FIFO replaces entry 0 with the 8th instruction because the previous instruction has been placed in entry 3. FIFO misses a one-hit chance when it processes the 10th instruction. Even though an entry is frequently accessed, the FIFO always replaces it based on the replacement order. On the other hand, Hitpoint replaced entry 2 with the 8th instruction because the 7th instruction was matched to entry 1 and the next replacement point has been moved to entry 2. The 7th instruction is not exactly matched to the 1st instruction, but their addresses are neighbored and thus share one entry. The Hitpoint algorithm maintains the frequently used special entry and thus benefits from temporal locality.

#### IV. EXPERIMENTS

##### A. Evaluation

Based on the design presented in Section III, the 4-entry WLSB and 8-entry WLSB have been coded in VHDL, and Cadence NC\_VHDL was used for simulation. We used the SPEC2K applications and simulated 1,000,000 instruction traces after the number of instructions according to SimPoints [2]. Only the memory instruction traces are injected to the WLSB while maintaining the appropriate time between successive memory instructions. The SPECK2K applications are compiled with the gcc compiler using -O3 optimizations. It should be noted that the results in this paper correspond to only scalar instructions as generated by the gcc compiler although the WLSB is designed to work with WideWord instructions as well as scalar instructions. We leave the WideWord analysis for future work.

After the simulation of SPEC2K with 3 different replacement algorithms while all other configurations are set as described in Section II, the dedicated load replacement did not perform much better than FIFO. Only 0.02% increase on average for 11 programs was detected. The WLSB has been filled up by load instructions most of the time. It is rare to have consecutive store instructions which are independent from previous instructions. Therefore, a situation where all the entries are filled up by only store instructions is unusual. So although our intuition told us otherwise, there is no need to maintain a dedicated entry for loads only because the

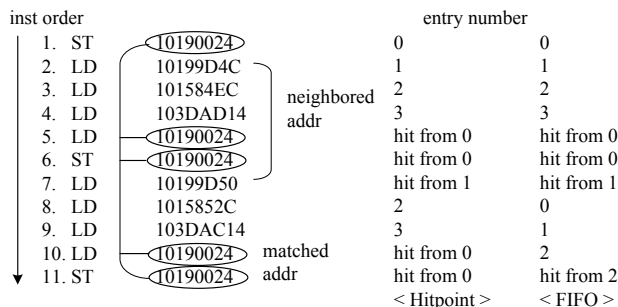


Figure 3. Comparison of two replacement algorithms for SPEC2K art

situation where stores fully occupy the WLSB rarely occurs, at least for the benchmarks we simulated. However, Hitpoint replacement showed a 2.54% load hit increase on average. Especially, it provided a great benefit to art which has a characteristic that loads and stores the same address every 3 or 4 memory instructions. It is crucial to keep such special memory data retained in the WLSB as long as possible.

When it is assumed that the 4-entry WLSB is used only with the conventional scalar instruction set architecture, it can be easily compared to a 32-entry scalar load/store buffer (SLSB) with a 4B dataline because it has the same number of register bits from the perspective of data buffer size. Both require 1024 bits of register space; they simply differ in their organization of these bits.

Fig. 4 shows a comparison of load hit rates for a 32-entry SLSB (FIFO) and 4-entry WLSB (Hitpoint). The SLSB has been designed to have full associativity to maximize the hit rate, and the FIFO replacement algorithm is adopted to minimize access latency. As can be seen from Fig. 4, the 4-entry WLSB with Hitpoint has 5.72% better performance on average than the 32-entry SLSB. 7 SPEC2K benchmarks out of 11 derive benefits from preloading, not just from the number of entries. Even though art is excluded from the comparison by its unique characteristic, the 4-entry WLSB is still 3.60% better than the 32-entry SLSB. Especially for SPEC2K FP benchmarks, which are mesa, art, equake and apsi, while the 32-entry SLSB achieved a 40.70% load hit rate, the 4-entry WLSB exhibited a 60.97% load hit. This is a significant improvement and demonstrates that the 4-entry WLSB can be exploited more in programs which exhibit spatial locality like the SPEC2K FP programs. In addition, the 4-entry WLSB has advantages in terms of access latency and area. Although fully associative caches yield higher hit rates, they also manifest the characteristic that search time increases as the number of entries increases. The 32-entry fully associative SLSB will simply be much slower than a 4-entry alternative. Set associativity can be adopted to mitigate this penalty however this impacts performance. Furthermore, the area needed for more tags causes the overall area of the 32-entry SLSB to be greater than that of the 4-entry WLSB.

Fig. 5 shows the load hit comparison among 4-entry WLSB (Hitpoint), 8-entry WLSB (FIFO) and 8-entry WLSB (Hitpoint). 8-entry WLSB (Hitpoint) is 0.81% better than 8-WLSB (FIFO) and 10.69% better than 4 WLSB (Hitpoint). This is a 16.41% increase over the 32-entry SLSB (FIFO).

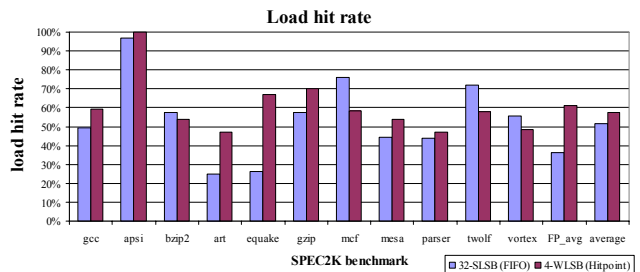


Figure 4. Load hit rate, 32-SLSB (FIFO) vs 4-WLSB (Hitpoint)

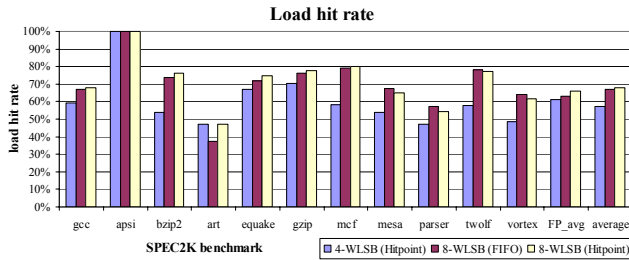


Figure 5. Load hit rate among 4-WLSB (Hitpoint), 8-WLSB (FIFO) and 8-WLSB (Hitpoint)

The 8-entry WLSB (FIFO) suffers from poor performance for art, but it shows fine performance in the rest of the benchmarks. If only the FP benchmarks (art, mesa, apsi and equake) are considered, the 4-entry WLSB shows a load hit rate that is close to that of the 8-entry WLSB. Therefore, the 4-entry WLSB is a good option for high-speed FP systems, given its smaller area.

### B. Implementation

The specification of the WLSB required on the order of 2000 lines of RTL-level behavioral VHDL code. The VHDL was synthesized using Synopsys Design Compiler and targeting IBM 90nm technology. Fig. 6 shows synthesis results for the 32-entry fully associative SLSB (FIFO), 4-entry WLSB (Hitpoint) and, 8-entry WLSB (Hitpoint) and including total area and dynamic power consumption corresponding to differing clock period targets. The 4-entry WLSB has been synthesized with latencies up to 1.6ns; 8-entry WLSB, 1.9 ns; and 32-entry SSB, 2.2ns. From the Figure, the 4-entry and 8-entry WLSBs can achieve greater speeds than the 32-entry SLSB. Also, the 4-entry WLSB occupies a much smaller area than the 32-entry SLSB with the same clock period. The 32-entry SLSB suffers from search time by the nature of full associativity, while the control logic is simpler than others. There are always trade-offs between speed and miss rate in cache design, and usually different cache organizations are adopted depending on system requirements. However, the 4-entry and 8-entry WLSBs break through the speed vs. miss rate trade-off. In 2.4 ns, the 32-entry SLSB exhibits very low dynamic power because if an efficient search time is guaranteed, most of the dynamic power is consumed by registers. It has a very simple control logic by supporting only one instruction type. On the other hand, the 4-entry and 8-entry WLSBs do not have inherent features which can benefit from non-stringent clock constraints. This is mainly due to the complicated control logic needed to support both instruction types.

## V. CONCLUSION

This paper introduced a WLSB that has been specifically designed to benefit the DIVA PIM but is expected to be useful for other embedded processing platforms. It achieves significant performance improvements while occupying only

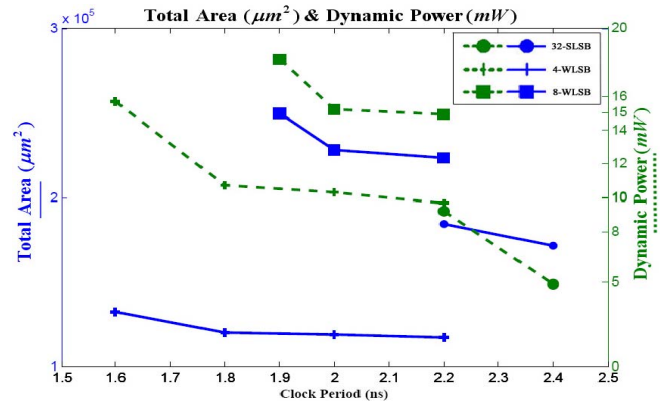


Figure 6. Total Area and Dynamic Power

a small area. A 4-entry WLSB has been compared to an alternative 32-entry SLSB which contains the same number of data storage bits, and the WLSB has been shown to be superior with regard to every performance metric. It is especially noteworthy that the 4-entry WLSB appears to defy the common sense that a smaller number of cache entries results in higher miss rates, while showing a noticeable load hit rate on SPEC2K FP benchmarks. The WLSB was synthesized in IBM 90 nm technology, and the resulting implementation occupies less than a seventh of a square mm and is projected to run at 1.6ns cycle time with 15.72mW of dynamic power dissipation.

## REFERENCES

- [1] Jeffrey Draper et al. "The Architecture of the DIVA Processing-In-Memory Chip." In *Proceedings of the ACM International Conference on Supercomputing*, June 2002.
- [2] Brad Calder, Glenn Reinman. "A Comparative Survey of Load Speculation Architectures." *Journal of Instruction-Level Parallelism 1*, May 2000.
- [3] Standard Performance Evaluation Corporation, <http://www.specbench.org>
- [4] Manfred Schlett. "Trends in Embedded Microprocessor Design." *Computer*. Vol. 31. Aug 1998.
- [5] J. Hennessy and D. Patterson. "Computer Architecture: A Quantitative Approach." Morgan Kaufman, 3<sup>rd</sup> edition, 2002.
- [6] Dan Nicolaescu et al. "Caching Values in the Load Store Queue." In *Proceedings of The IEEE Computer Society's 12<sup>th</sup> Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, Oct. 2004.
- [7] I. Park, C. L. Ooi, and T. N. Vijaykumar. "Reducing Design Complexity of the Load/Store Queue." In *Proceedings of the 36<sup>th</sup> International Symposium on Microarchitecture*, Dec. 2003.
- [8] Sam S. Stone et al. "Address-Indexed Memory Disambiguation and Store-to-Load Forwarding." In *Proceedings of the 38<sup>th</sup> International Symposium on Microarchitecture*, Nov. 2005.
- [9] T. Sherwood, E. Perelman, G. Hamerly and B. Calder. "Automatically Characterizing Large Scale Program Behavior." In *Proceedings of the 10<sup>th</sup> International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.
- [10] Moinuddin K. Qureshi et al. "The V-Way Cache : Demand-Based Associativity via Global Replacement." In *Proceedings of the 32<sup>nd</sup> International Symposium on Computer Architecture*, June 2005.