# A Memory Rename Table to Reduce Energy and Improve Performance

Joseph Pusdesris, Benjamin VanderSloot, and Trevor Mudge
University of Michigan
Ann Arbor, MI
{joemp,benvds,tnm}@umich.edu

## ABSTRACT

A memory rename table for improved performance, reduced complexity, and reduced energy consumption is proposed and evaluated. It gives an average 8.7% speedup and 7.9% reduction in core and cache energy. The evaluation employs a simulation model for an out-of-order core, similar to the ARM Cortex A15, and McPAT for energy measurements. The improvements are the result of filtering nearly half (45.4%) of memory accesses before they go to cache. Complexity is reduced by replacement of a load store queue with a scalable renaming system. The changes are transparent to the ISA.

## Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles—pipeline processors

## Keywords

Load store unit; memory system; energy-efficiency

## 1. INTRODUCTION

To cope with the high cost of memory access, modern architectures provide a large number of general purpose registers. These registers offer a dense set of short term storage within the CPU to avoid accessing memory. Unfortunately, short term values cannot always take advantage of these registers. There are situations known to cause this behavior, for example:

- Register pressure cause values to be spilled and filled from memory.
- Registers must be demoted across function calls.
- Compilers are not able to disambiguate pointers and conservatively keep the values in memory to guarantee correctness.


*ISLPED'14,* August 11–13, 2014, La Jolla, CA, USA.

http://dx.doi.org/10.1145/2627369.2627655 .

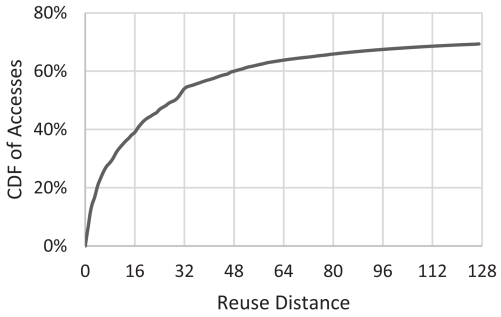


**Figure 1: A CDF that depicts the locality in SPECINT when compiled for the ARM ISA. The reuse distance is measured in number of distinct addresses accessed between repeated accesses. Despite the reuse distance being low, these values are not promoted to registers by software.**

While a variety of techniques to reduce these restrictions have been proposed, they have not seen widespread adoption. This is likely because of the required changes to the programming interface. The most common architectural approach used in modern out-of-order processors is not to prevent the situations listed above, but instead to speed up the short term spills via a sophisticated load-store-unit (LSU) in conjunction with a high-bandwidth, low-latency L1 cache.

To show just how prevelent these sort term spills are, we will use a metric called reuse distance. Reuse distance is the distance between accesses to the same address; the distance here is measured in number of accesses to distinct addresses. Figure 1 quantifies the reuse distance present in SPECINT. This figure shows that if we can preserve 64 of the most recently accessed values they will satisfy more than 60% of future memory accesses.

This reuse distance is so small that the L1 cache may be over equiped for the general case. Since a small reuse distance has such a high coverage, it stands to reason that a structure sized hundreds of bytes could be used instead. In traditional designs, the closest structure to that size is the store buffer within the LSU. However, since the store buffer often only holds pre-retirement stores, and does not include post-retirement stores or any loads, the values held in this structure are a small percentage of the total accesses. The low coverage of this structure is reflected by a low rate of forwarding. Infrequent forwarding motivated designers to access the data cache in parallel with the store buffer. While this is necessary to provide the lowest average latency, this

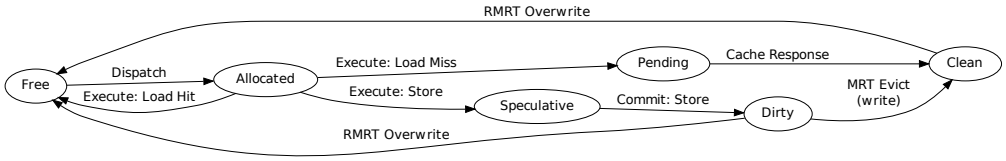


**Figure 2: The FSM associated with each register in the MRF. A register is marked as allocated when a memory operation is dispatched. When the operation is executed and accesses the MRT, the register is either marked free, pending, or speculative. The registers will later be updated to Dirty and Clean, eventually being marked free when the entry is no longer referenced by the RMRT.**

is unnecessarily taxing on power resources when the LSU services the request by forwarding.

There have been many LSUs implemented and proposed with differing capabilities. The baseline in this work is an LSU which restricts issue ordering such that accesses cannot pass stores to the same address. Store-to-load forwarding is also included in the baseline. These are the optimizations to be expected from an ARM Cortex A15-like core. More aggressive cores may speculatively issue with respect to memory ordering.

While effective, many LSU designs are expensive, featuring comparator matrices, storage registers, ordering logic, scheduling logic, and requiring the L1 cache to be multi-ported with low latency. Our design offers significant reduction in energy consumption and design complexity compared to the equivalent level of optimization in a traditional LSU. This increase in scalability will enable more aggressive optimizations which would otherwise be unattainable within the same resource budget.

An alternative to an LSU-centric approach is an L0 cache, but the L0 cache has several shortcomings. A granularity of block size instead of word size is wasteful in such a small structure. Additionally, an L0 cache does not support the storage of speculative data. Our design and traditional LSU can contain multiple uncommitted stores, unlike an L0.

# 2. DESIGN

Our design is intended to reduce energy consumption for out-of-order cores while requiring no modification to the ISA or programming model. With our design many memory accesses which would traditionally be L1 data cache hits no longer access the cache at all, and are instead forwarded from an earlier memory access.

Several decisions were made specifically to reduce energy consumption. One such decision is to make the design non-speculative, requiring that correct memory ordering is enforced at execution. Many other LSU designs will issue memory operations in a speculative order and squash on detection of an incorrect prediction. By avoiding this behavior, there is no longer wasted execution that gets squashed, misspeculation detection circuitry, or prediction logic. We also reduce the number of comparators to a small constant number, relative to the LSQ size. Traditional LSQ designs often need N, or even $N^2$ comparisons as they are fully associative, comparing every addresses to every other address either at issue or execute.

Our design is similar to a register renaming pipeline. Just as logical registers are translated to physical register indices in an out-of-order pipeline, memory addresses are translated to register file indices. These translations may index the same register file as the register rename table, but for the remainder of this paper they will refer to a separate register file, called the *Memory Register File (MRF)*. Each entry in this MRF has associated state bits. Similar to the freelist in register renaming, this state is used to track when an entry is *Free* to be allocated, if an entry can be made *Free*, and what to do with the entry at different stages in the pipeline. The state machine for this tracking is shown in Figure 2.

The active translations of memory address to MRF entries are stored in a 4-way set associative structure called the *Memory Rename Table (MRT)*. This structure is similar to what was used for register renaming in VCA [5]. The pipeline must contain two of these MRTs. One MRT, accessed in the execute stage, contains a valid picture of memory at issue time, including the effect of the stores which have not yet been committed. The other MRT only contains committed information and it is accessed in the retire stage of the pipeline. This secondary MRT is used to recover from a branch misprediction and will be referred to as the *Retirement Memory Rename Table (RMRT)*.

The above components are placed into an out-of-order pipeline as shown in Figure 3. Unlike register renaming, the renaming done by our design happens in the execute stage, out-of-order. This is problematic as renaming in the order in which addresses are produced leads to incorrect execution. To prevent this, the address queue will restrict the order in which memory operations can issue.

## 2.1 Issue stage

The address queue, pictured in Figure 4, imposes the following restrictions for memory operations to issue:

- Stores cannot pass more than N other stores, where N is the associativity of the MRT.
- Accesses cannot pass stores to the same address.
- Restrictions imposed by the consistency model of the implemented ISA.

These restrictions are enforced by allowing execution of only the N oldest non-executed memory operations. The younger operations are in a FIFO buffer until space is available in the N issue slots. Note that the second restriction requires that for a load to proceed, its address must be resolved as well as any stores that precede it because of our lack of speculation.

## 2.2 Execution stage

Once issued, the address of a memory operation is searched in the MRT. If the instruction is a load and a hit in the MRT, the value is read from the memory register referenced by the MRT entry and releases the register which was allocated in

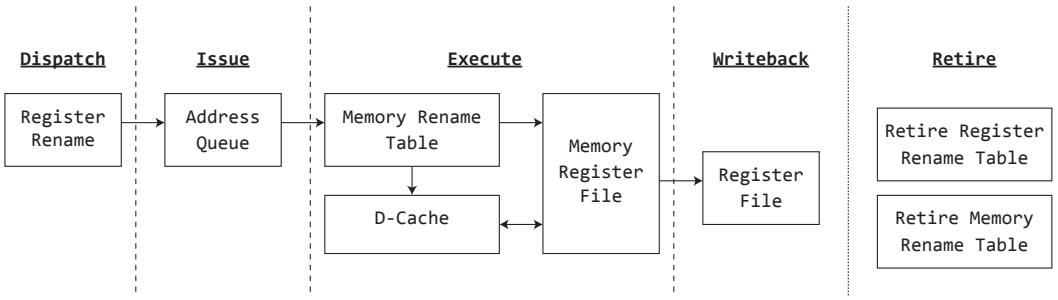


**Figure 3: Our load-store unit's components placed into the proper stages of a pipeline. Only the components that are part of or interface directly with our design are depicted.**

dispatch to the *Free* state. If the load's address misses in the MRT, a cache access is issued and the result is stored into the *Allocated* Memory Register. An MRT entry is also allocated, evicting the least recently used non-*Speculative* entry in that set. The address and MRF number are then written to that MRT entry and the memory register is transitioned to the *Pending* state, where it remains until it is transitioned to the *Clean* state once the cache responds. If the evicted MRT entry's register is in the *Dirty* state, indicating a store has not written back to the L1 cache, then a cache write is issued.

The net effect of the above is that the MRT filters loads before accessing the cache. This causes the cache latency to increase relative to a traditional design where the LSU and cache are accessed in parallel, but the hit rate is typically high enough that the lower latency of MRT forwarding compensates for the higher latency of the cache. The MRT is expected to have significantly higher hitrates than the store buffer in a traditional LSU since it includes values from retired instructions, is larger, includes loads, and has an even larger effective capacity since duplicates do not use additional entries.

If the issued operation is a store to an address that hit in the MRT, then the MRT entry's Memory Register is updated to the register allocated for this instruction at dispatch and marked as *Dirty*. The address component is left unchanged since it is already the address that hit, but the Memory Register Number is overwritten with the new one.

For a store, an MRT miss behaves similarly to an MRT hit, except both the address and Memory Register Number are written to an evicted MRT entry. The MRT eviction is done in the same way described for a load miss.

If all MRT entries for a set were allowed to be in the *Speculative* state a deadlock could occur. The address queue prevents this by only allowing ops to pass N stores. If this were not limited, then the N subsequent stores could execute, and therefore hold on to the entire block of entries in the MRT, potentially preventing forward progress.

## 2.3 Retirement stage

At retirement, memory instructions write their entry changes to the RMRT, similar to how they wrote to the MRT at execute. Since it is already known which entry number will be written, no search is needed and the RMRT can be direct mapped. The write of an RMRT entry may overwrite an older entry. In this event, the overwritten entry's register is moved to the *Free* state. In addition to this, if the instruction is a store, it must update its register's state to *Dirty*. Dirty values are not written to the cache until they

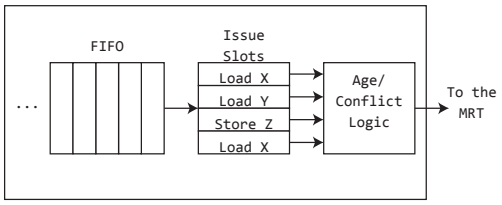


**Figure 4: The address queue of our design. Instructions fill the 4 Issue Slots where they are compared to verify which are allowed to proceed. If these slots fill, then the FIFO buffer will allow the previous pipeline stages to continue execution.**

are evicted from the MRT. This significantly reduces the number of stores that take place, since only the last value written to an address while it is resident in the LSU, the one evicted from the MRT, will be written. In the event of a squash, the RMRT contents are copied into the MRT, immediately returning the MRT to a valid state for continued execution.

# 3. METHODOLOGY

Trace-based simulation was used to evaluate the advantages of our design. This in-house simulator implements a superscalar pipeline with drop-in load-store units featuring our LSU and the LSU discussed in Section 1 as baseline.

The simulator was fed with an oracle dynamic instruction stream. This model features no instruction fetch side hazards such as branch prediction or instruction cache misses. This was considered fair since both pipelines employ near identical branch resolution mechanisms and penalty. The results should only bias in favor of the baseline because due to the RMRT, the MRT will be warm on recovery and the store buffer will not. This oracle trace was generated using gem5 [2] with the ARM instruction set.

The benchmarks used consisted of all of SPECINT 2006 [3]. Each trace includes 2 billion instructions, but no statistics are collected for the first 1 billion to allow for cache warm up. The traces were fed into each architecture model for detailed comparisons.

To monitor detailed performance and energy cost, the simulator keeps track of a variety of statistics including register file, cache, and memory read/write as well as cycle count. The simulator was configured to closely match the ARM cortex A15 [1].

We used McPat to estimate power and energy[4]. For these studies we targeted a 22nm technology node, modeling

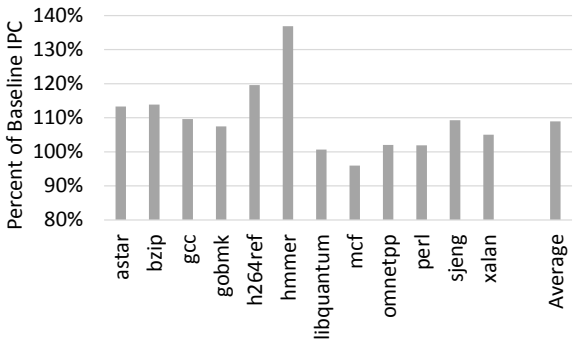


**Figure 5: A detailed per-benchmark graph of the IPC reported relative to the store buffer baseline. The MRT is 4-way set associative with 64 entries.**

multi-gate devices and aggressive interconnect projections on wire technologies. The nominal clock frequency was set to 1.7 GHz. This value matches the clock frequency of current industrial design, and we validated the peak power figures obtained with our model against publicly available data of comparable commercial microprocessors.

# 4. EVALUATION

Tests are performed across the SPECINT benchmarks using a configuration comparable to an ARM Cortex A15. The experiments performed give a sense of the energy efficiency and IPC gained by using our LSU. All values are presented relative to the baseline.

## 4.1 IPC

While our design increases L1 latency by deferring access until an MRT miss is confirmed, there is typically still a net performance gain. The performance of our design across SPECINT demonstrates an average increase in IPC of 8.7% over the baseline. The cause of this speedup is twofold. First, the LSU access is modeled as one cycle faster than a typical L1 access. This lets a dependent instruction proceed a cycle sooner. Second, this filtering of accesses will reduce contention on the L1 cache port(s) which would otherwise cause stalls. The per-benchmark speedup is shown in Figure 5.

## 4.2 Cache Bandwidth

A summary of average reduction in cache accesses is shown in Figure 6. In total, 45.4% of memory accesses are removed, from both the reads and writes relative to the baseline execution. Optimizations in the design such as post-retirement storage, store coalescing, load-to-load forwarding, and store-to-load forwarding are responsible for these results.

## 4.3 Power and Energy

We first evaluated the impact of our architecture on the power consumption of the data cache using McPAT. Our results report that we reduce data cache dynamic power by 42% on average. On top of this, the IPC is increased so leakage energy of the data cache per operation is reduced as a result, by 9.5%.

We then evaluated the total power and energy consumed by a core augmented with our LSU. Despite the reduction in data cache power consumption, our studies reported a slight increase in total core power consumption: 2.6%. We


**Figure 6: The reduction in memory access as provided by our Load-Store Unit with a 64 entry, 4-way set associative MRT. Significant decrease is demonstrated both in the number of reads and writes performed when compared to the baseline store buffer.**

justify these results with the higher IPC achieved by our design. While our design reduces the burden and the power consumed by the memory subsystem, it increases the load on the rest of the pipeline by increasing the IPC. As a result, our design improves core energy efficiency, yielding a significant reduction in the energy needed to execute a task. Our results show that a core deploying our load store unit achieves average energy savings of 7.9%.

# 5. CONCLUSION

As an artifact of the way software is written or generated today, the reuse distance of many values in memory is very short. Short enough to be stored in a physical register file instead of the cache. To exploit this attribute, we present a LSU that keeps most recently used memory addresses in a register file managed by a pair of set associative rename tables.

Our work supports memory optimizations at very little power cost. It has post-retirement storage, store coalescing, store-to-load, and load-to-load forwarding. With these optimizations, we can prevent 45% of cache accesses, reducing cache power. Accessing these structures sequentially increases the latency of a cache access, but this is overcome by the high hit rate in the low latency LSU and reduction in cache port contention. The net result is an 8.7% speedup and a 7.9% energy reduction.

# 6. REFERENCES

[1] ARM. *Cortex-A15 MPCore Technical Reference Manual, Revision: r3p3.*

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[3] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[4] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480, 2009.

[5] D. W. Oehmke, N. L. Binkert, T. Mudge, and S. K. Reinhardt. How to fake 1000 registers. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pages 7–18. IEEE Computer Society, 2005.